User Manual

Release 09/13, Revision 30.156

TESSY Version 3.0

Razorcat Development GmbH

Imprint

Razorcat Development GmbH
Witzlebenplatz 4
Germany, 14057 Berlin
tel: +49 (030) 53 63 57 0
fax: +49 (030) 53 63 57 60
e-mail: support@razorcat.com
internet: http://www.razorcat.com

Windows is a registered trademark of Microsoft. TESSY and CTE are registered trademarks of Razorcat Development GmbH.
All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same is claimed.

Liability exclusion

Razorcat Development GmbH assumes no liability for damage that is caused by improper installation or improper use of the software or the non-observance of the handling instructions described in this manual.

Thanks

Various contents are based on application notes and publications on TESSY written by Frank Büchner, Hitex Development Tools GmbH. We would like to thank Frank for his valuable contribution and commitment in supporting TESSY and spotlighting functionalities and features.

Contents

Preface **xv**
About TESSY . . . xvi
How to use this manual . . . xvii
Subject matter . . . xvii
Helpers . . . xix
Safety procedures . . . xxi
Core workflow and registration for safety information . . . xxi
Verification and certification of TESSY . . . xxii
Instrumentation for coverage measurement . . . xxii
Adaptation to target environment . . . xxiii
Operating limits . . . xxiii
New features in TESSY 3.0 . . . xxiv
Revised graphical user interface (GUI) . . . xxiv
Improved main perspective . . . xxiv
Traceability of requirements . . . xxiv
Enhanced coverage handling . . . xxv
Improved integration of the Classification Tree Editor (CTE) . . . xxv
Tabular Test Data Editor (TDE) . . . xxvi
Enhanced Usercode editor . . . xxvi
Improved test data plots . . . xxvi
Enhanced component testing . . . xxvii
Application Programming Interface (API) and commandline interface . . . xxvii
Troubleshooting . . . xxviii
Creating the TESSY Support File . . . xxviii
Tipps for a better TESSY Support File . . . xxx

1 Installation and registration **1**
1.1 Technical requirements . . . 1
1.2 Setup . . . 1
1.3 Registration . . . 6
1.3.1 The License Manager . . . 7
1.3.2 Solving problems . . . 8
1.4 Uninstallation . . . 9

2 Migrating from TESSY 2.x to 3.x **11**
2.1 Changes in handling . . . 11
2.2 Importing previous projects . . . 13
2.3 Missing features . . . 15
2.3.1 Python based report . . . 15
2.3.2 THAI . . . 16
2.3.3 OBT . . . 16
2.3.4 Alias names . . . 16
2.3.5 Debugger Scripting Prolog / Epilog . . . 16
2.3.6 Generation of test steps . . . 16
2.3.7 Diff Viewer . . . 16

3 Basic handling **17**
3.1 Creating databases and working with the file system . . . 18
3.1.1 Creating a project database . . . 18
3.1.2 Optional: Using a specific environment setting . . . 22
Creating a configuration file . . . 23
3.1.3 Deleting a project . . . 23
3.2 Understanding the graphical user interface . . . 25
3.2.1 Menu bar . . . 25
3.2.2 Tool bar . . . 26
3.2.3 Perspectives and perspective (tool) bar . . . 26
3.2.4 Views . . . 27
3.2.5 Status bar . . . 30
3.3 Using the context menu and shortcuts . . . 31
3.3.1 Context menu . . . 31
3.3.2 Shortcuts . . . 31

4 Basic knowledge **33**
4.1 Unit testing of embedded software . . . 34
4.1.1 Standards that require testing . . . 34
4.1.2 About unit testing . . . 35
What is unit testing? . . . 35
What are the benefits? . . . 35
4.1.3 Considerations for unit testing . . . 36
Which units are good test candidates? . . . 36
What is not in the scope of unit testing? . . . 36
Why is regression testing necessary? . . . 37
Who should conduct the tests? . . . 37
What is special for testing embedded software? . . . 38
4.1.4 Methods for unit testing . . . 38
a. Test application . . . 38

b. Original binary test . . . 39
Pros and cons . . . 39
4.1.5 Conclusion . . . 40
4.2 The Classification Tree Method (CTM) . . . 41
4.2.1 General . . . 41
4.2.2 Steps to take . . . 42
4.2.3 Example is_value_in_range . . . 48
Test-relevant aspects . . . 48
Forming classes . . . 49
A first range specification . . . 49
A second range specification . . . 50
Extending the tree by a boundary class . . . 51
Another interesting test case specification . . . 52
The completed classification tree . . . 53
The completed test case specification . . . 55
Another test case specification . . . 56

5 Practical exercises **59**
5.1 Quickstart 1: Exercise is_value_in_range . . . 61
5.1.1 Creating a test project . . . 62
5.1.2 Specifying the target environment . . . 64
5.1.3 Adding the test object and analyzing the C-source file . . . 65
5.1.4 Editing the test object interface . . . 68
5.1.5 Designing test cases . . . 69
5.1.6 Adding test cases and test steps . . . 69
5.1.7 Entering test data . . . 71
5.1.8 Executing the test . . . 74
5.1.9 Repeating the test run with coverage instrumentation . . . 75
5.1.10 Analyzing the coverage . . . 76
The flow chart view . . . 77
The Branch (C1) Coverage view . . . 77
The MC/DC Coverage view . . . 77
Analyzing . . . 78
5.1.11 Creating a report . . . 81
5.1.12 Repeating the test run with requirements . . . 83
Importing requirements . . . 83
Committing the requirements document . . . 85
Linking test cases with requirements . . . 86
Creating a planning coverage report . . . 87
Executing the test and examining the coverage . . . 90
5.1.13 Reusing a test object with a changed interface . . . 93
Changing the interface of the test object . . . 93

Assigning the changed interface of the test object 98
5.2 Quickstart 2: Exercise Classification Tree Editor (CTE) 101
5.2.1 Creating classifications 102
5.2.2 Creating classes 104
5.2.3 Entering test data 104
5.2.4 Creating test cases 107

6 Working with TESSY **113**
6.1 Overview perspective: Organizing the test 117
6.1.1 Structure of the Overview perspective 117
6.1.2 Test Project view 118
Icons of the view tool bar 119
Status indicators 119
Creating tests 120
Analyzing modules 121
Executing tests 122
Instrumentation settings 124
Viewing test results 125
Creating reports 126
Batch test operations 130
Importing and Exporting 132
6.1.3 Properties view 134
General 135
Sources: Sources pane 136
Sources: Compiler pane 137
Coverage 140
Attributes 141
Specification / Description / Comment 142
6.1.4 Requirements Coverage view 143
6.1.5 Test Items view 143
Icons of the view tool bar 144
Status indicators 144
Creating test cases and test steps 145
6.1.6 Test Results view 147
6.1.7 Evaluation Macros view 147
6.1.8 Console view 148
6.1.9 Suspicious Elements view 149
6.2 Import perspective: Importing requirements 150
6.2.1 Importing requirements 150
6.2.2 Formats 152
6.3 Requirement management 153
6.3.1 Structure of the Requirement Management perspective 154

6.3.2 RQMT Explorer view . . . 155
Icons of the view tool bar . . . 156
Status indicators . . . 156
Renaming / Alias . . . 156
Committing requirements . . . 157
Creating requirements . . . 158
Editing requirements . . . 159
6.3.3 Requirements List view . . . 160
Icons of the view tool bar . . . 160
Viewing requirements, versions, IDs . . . 161
6.3.4 Test Means view . . . 162
Icons of the view tool bar . . . 162
6.3.5 VxV Matrix view . . . 163
Icons of the view tool bar . . . 163
6.3.6 Link Matrix view . . . 164
Icons of the view tool bar . . . 164
Status indicators . . . 164
Adding and removing elements . . . 165
Updating requirement links . . . 166
6.3.7 Suspicious Elements view . . . 167
Icons of the view tool bar . . . 167
Determine changes that caused suspicious status . . . 168
6.3.8 Attached Files view . . . 170
Icons of the view tool bar . . . 170
6.3.9 Attributes view . . . 171
Icons of the view tool bar . . . 171
6.3.10 History view . . . 173
Icons of the view tool bar . . . 173
Reviewing changes . . . 173
6.3.11 Related Elements view . . . 175
6.3.12 Requirements Coverage view . . . 175
Icons of the view tool bar . . . 176
Planning tab . . . 177
Execution tab . . . 178
6.4 TEE: Configurating the test environment . . . 179
6.4.1 Starting the TEE . . . 180
6.4.2 Structure and handling of the TEE . . . 180
6.4.3 Status indicators . . . 182
6.4.4 Configuration files . . . 184
6.4.5 Using default configurations . . . 185
6.4.6 Enable configurations . . . 186
Enable compiler and target in paragraph General . . . 186

Enable configuration within paragraph System . . . 187
6.4.7 Customizing enabled configurations . . . 188
Adding and editing attributes . . . 188
6.5 TIE: Preparing the test interface . . . 191
6.5.1 Structure of the TIE perspective . . . 192
6.5.2 Test Project view . . . 192
6.5.3 Properties view . . . 192
6.5.4 Interface view . . . 193
Handling . . . 193
Icons of the view tool bar . . . 194
Interface elements . . . 194
Setting passing directions . . . 195
Setting the data format . . . 197
Setting passing direction of special data types . . . 198
Defining stubs for functions . . . 198
Other interface settings . . . 200
6.5.5 Plots view . . . 201
6.6 CTE: Designing the test cases . . . 202
6.6.1 The basic idea . . . 202
6.6.2 Starting CTE . . . 202
6.6.3 Structure of the CTE perspective . . . 203
6.6.4 Test Project view . . . 203
6.6.5 Properties view . . . 203
6.6.6 Classification Tree view . . . 204
Icons of the view tool bar . . . 204
Structure of the Classification Tree view . . . 205
Designing a classification tree . . . 205
Creating classifications and classes . . . 206
Creating test cases . . . 207
6.6.7 Test Data view . . . 209
6.7 TDE: Entering test data . . . 210
6.7.1 Structure of the TDE perspective . . . 210
6.7.2 Test Project view . . . 211
6.7.3 Test Results view . . . 211
6.7.4 Evaluation Macros view . . . 212
6.7.5 Test Items view . . . 212
6.7.6 Properties view . . . 212
6.7.7 Test Data view . . . 213
Icons of the view tool bar . . . 213
Viewing the interface of the test object . . . 214
Entering values . . . 215
Entering evaluation modes . . . 220

Ignoring values for a test step . . . 222
Pointers . . . 222
Static local variables . . . 223
6.7.8 Test Definition view . . . 224
6.7.9 Prolog/Epilog view . . . 225
Entering C code . . . 226
Using evaluation macros . . . 228
6.7.10 Usercode Outline view . . . 230
6.7.11 Plots view . . . 231
Icons of the view tool bar . . . 231
Entering plots . . . 231
6.8 CV: Analyzing the Coverage . . . 236
6.8.1 Structure of the CV perspective . . . 238
6.8.2 Test Project view . . . 238
6.8.3 Called Functions view / Code view . . . 238
6.8.4 Flow Chart view . . . 239
Icons of the view tool bar . . . 239
Viewing functions . . . 240
Selecting Elements . . . 241
Searching for uncovered decisions and unreached branches . . . 242
6.8.5 Branch (C1) Coverage view . . . 243
Coverage percentage . . . 243
6.8.6 MC/DC Coverage view . . . 244
Selecting Decisions . . . 244
6.8.7 MCC Coverage view . . . 245
6.8.8 Coverage Report views . . . 245
6.9 IDA: Assigning interface data . . . 246
6.9.1 Structure of the IDA perspective . . . 247
6.9.2 Test Project view . . . 247
6.9.3 Properties view . . . 248
6.9.4 Compare view . . . 248
Status indicators . . . 248
Comparing interfaces and assigning changes . . . 249
6.10 SCE: Component testing . . . 251
6.10.1 Creating component tests . . . 252
6.10.2 Preparing the test interface . . . 253
6.10.3 Designing the test cases . . . 254
6.11 Backup, restore, version control . . . 255
6.11.1 Backup . . . 255
6.11.2 Restore . . . 256
Restore into the original location . . . 256
Restore into another location . . . 256

6.11.3 Version control 257
6.12 Command line interface 258
6.12.1 Invoking tessycmd.exe 258
6.12.2 Usage of tessycmd.exe 259
6.12.3 Commands 259
6.12.4 Example: DOS script 260

Appendix **262**

A Abbreviations **263**

B Glossary **264**

C List of Figures **268**

D List of Tables **273**

Index **275**

Preface

About TESSY xvi
How to use this manual xvii
Subject matter xvii
Helpers xix
Safety procedures xxi
Core workflow and registration for safety information xxi
Verification and certification of TESSY xxii
Instrumentation for coverage measurement xxii
Adaptation to target environment xxiii
Operating limits xxiii
New features in TESSY 3.0 xxiv
Revised graphical user interface (GUI) xxiv
Improved main perspective xxiv
Traceability of requirements xxiv
Enhanced coverage handling xxv
Improved integration of the Classification Tree Editor (CTE) xxv
Tabular Test Data Editor (TDE) xxvi
Enhanced Usercode editor xxvi
Improved test data plots xxvi
Enhanced component testing xxvii
Application Programming Interface (API) and commandline interface xxvii
Troubleshooting xxviii
Creating the TESSY Support File xxviii
Tipps for a better TESSY Support File xxx

About TESSY

The test system TESSY was developed by the Research and Technology Group of Daimler. The former developers of the method and tool at Daimler were:
Klaus Grimm
Matthias Grochtmann
Roman Pitschinetz
Joachim Wegener

TESSY has been well-tried in practice at Daimler and is since applied successfully. TESSY is commercially available since spring 2000 and is further developed by Razorcat Development GmbH.

TESSY offers an integrated graphic user interface conducting you comfortably through the unit test. There are special tools for every testing activity as well as for all organizational and management tasks.

Dynamic testing is indispensable when testing a software system. Today, up to 80% of the development time and costs go into unit and integration testing. It is therefore of urgent necessity to automate testing processes in order to minimize required time and costs for developing high-quality products. The test system TESSY automates the whole test cycle; unit testing for programs in C/C++ is supported in all test phases. The system also takes care of the complete test organization as well as test management, including requirements coverage measurement and traceability.

How to use this manual

Apply for our e-mail list if you want to be informed of a new version of TESSY manual by sending an e-mail to support@razorcat.com.

Subject matter

The structure of the manual guides you through working with TESSY from the start to the specific activities possible. In order:

section	matter
Preface	Describes New features in TESSY 3.0 and contains important information about Safety procedures and Troubleshooting.
1 Installation and registration	Lists all technical requirements to work with TESSY and describes how to install the software.
2 Migrating from TESSY 2.x to 3.x	Lists the changed and new functions and handling within the new version. This will help you when switching from TESSY version 2.x to the new TESSY 3.x.
3 Basic handling	Explains the workflow of Creating databases and working with the file system. Check this section carefully to know how to handle your project data! The TESSY interface and basic handling is explained in the following sections Understanding the graphical user interface and Using the context menu and shortcuts.
4 Basic knowledge	Contains a brief introduction about unit testing with TESSY and the classification tree method.

continue next page

section	matter
5 Practical exercises	In this chapter you will get to know TESSY with the help of exercises that are prepared to follow easily though most functions of TESSY: The Quickstart 1: Exercise is_value_in_range is a very basic example to give you a fast introduction. The Quickstart 2: Exercise Classification Tree Editor (CTE) gives a short and easy introduction of handling with the Classification Tree Editor (CTE). It continues the Quickstart 1: Exercise is_value_in_range. We recommend strongly to check the practical exercises! It is very much easier to learn by doing than learning by just reading. Exercise first and then check the detailed information in the corresponding section of chapter "Working with TESSY".
6 Working with TESSY	**Explains in detail the unit test activities possible with TESSY.** As you will notice, the headlines of the sections follow the actions taken during a test. TESSY provides different editors and windows ("perspectives" and "views") for different configurations and steps taken during and after a test. You will find the name of the perspective or view as well as the description of the step within the headline, e.g. 6.6 CTE: Designing the test cases. Therefore, if you need help at some point, ask either "How do I handle . . . ?" or "Where am I?" and follow the headlines. **Important:** Read the Basic handling first, because basic functions of each editor are explained there.
Appendix	Contains a list of tables, a list of figures, a list of used abbreviations as well as definitions of used terms in the glossary. Check the glossary when you need some explanations of terms! The index in the very end of this manual provides the positions of your theme of interest.

Table 0.1: Where to find - matters of the several parts of the TESSY manual

Helpers

- The **Index** in the very end of this manual helps you by looking up a theme by a keyword.
- Various information is clearly represented within tables, e.g. icons and indicators (symbols of the interface) and their meanings. For a fast access to all tables consult the **List of Tables** in the appendix of this manual.
- Figures are used to demonstrate information described before. You may as well check the **List of Figures** to find those figures.
- All **cross references** as well as the content directory are active links (blue colored), so you can easily switch to the referenced chapter or section.

The sidearrow shows where to find information and references.

Orientation Note

Gray bordered explanatory boxes
provide further information and explanations for the respective issues and operations to be executed.

Font characters and signs

To help you to work with this manual, different font characters and signs are used to mark specific information:

what	for	example
→	instructions you are supposed to follow immediately	→ Open the TESSY interface.
>	navigation through a menue	→ Select "File" > "Open ..."
[...]	variable	→ Switch to "`[project root]\tessy`"
bold	accentuation	**Important:**
typewriter italic	input (you need to type this information) or output (message from system)	Enter ***`Test Example`***.
"`typewriter in quotes`"	path of data	"`C:\Program Files\Razorcat`"

continue next page

What	For	Example
"quotes"	indicate keys, buttons, etc.	→ Select "File" > "Open ..."
Ctrl+C	Keyboard characters are not marked in supposition they are commonly known.	Ctrl+C for pressing control and c

Table 0.2: Font characters

Important: You urgently need to know this for operating correctly!

Warning: There might be some damages to your data if you do not operate correctly! Please follow instructions carefully.

A light bulb provides hints, references and additional information on handling with TESSY for better usability.

Safety procedures

Core workflow and registration for safety information

Important: If you work with TESSY in a safety-relevant environment, please read this chapter carefully and register for our safety customer e-mail-list to be informed about known problems as described below!

TESSY can be used for testing of safety-relevant software. Therefore, based on our certified safety concept (certified for ISO 26262-08:2011 and IEC 61508-3:2010), TESSY development procedures are validated and verified, and our quality management assures the management of all development processes and constantly improves all procedures concerning quality and safety.

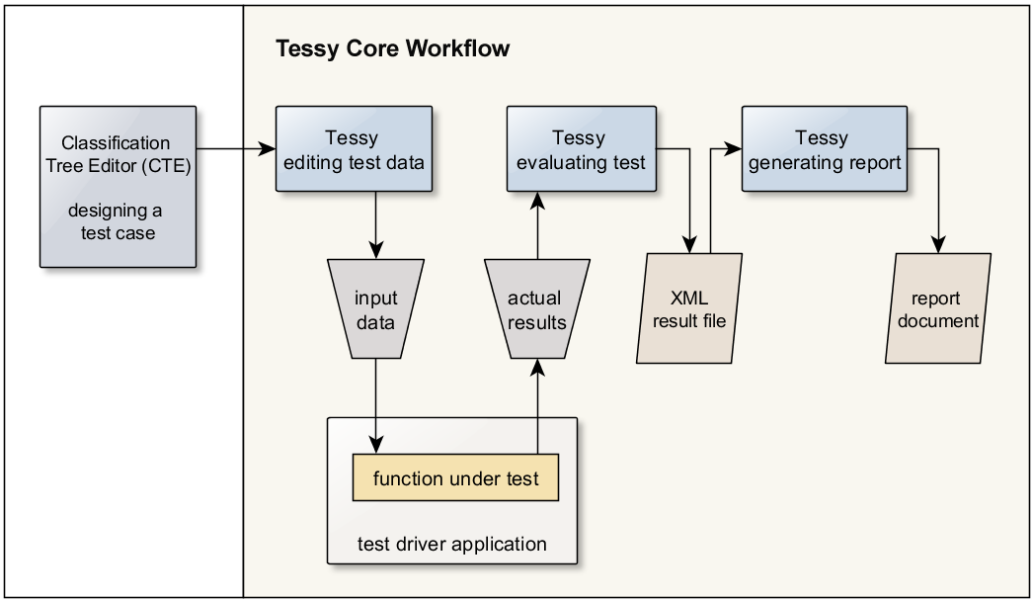


Figure 0.1: Core workflow of TESSY

The figure above shows the core workflow of TESSY that is fully automated and subject to tool qualification. All other tool capabilities like editing or environment and interface settings are additional features out of scope of the tool qualification. The core workflow of TESSY has been certified according to ISO 26262:2011 and IEC 61508:2010. Starting from editing of test data, the core workflow covers test execution, evaluation of test results and report generation. Additionally, the coverage measurements have been verified according to our certified safety plan. Please note, that the Classification Tree Editor

(CTE) which covers test preparation activities is not part of the certified core workflow of TESSY. Safety-relevant problems arising in released TESSY versions will be reported (once they are detected) and regarded closely to have them fixed as fast as possible. If you work with TESSY in a safety-related environment, **please register for our safety customer e-mail-list**:

- → Send an e-mail to support@razorcat.com
- → Topic: "Known problems requested"
- → Content: your contract data

You will be informed about current and newly arising "known problems" as well as workarounds.

Verification and certification of TESSY

The "Tool Qualification Pack" (TQP) is an additional purchase of documents and tests for TESSY, provided as baseline for the certification process in order to qualify TESSY as a software verification tool. Please contact via support@razorcat.com.

Instrumentation for coverage measurement

When executing tests using coverage measurements, it is recommended that all tests are executed once with and once without coverage instrumentation. TESSY uses a copy of the original source file when creating the test application. This copy of the source file will be instrumented for coverage measurements. Usually both test runs yield the same result, indicating that the instrumentation did not change the functional behavior of the test objects.
Please note, that the source code will be instrumented even if no coverage measurement has been selected in the following cases:

- When using the call trace feature
- When using static local variables

Some extra code will be added at the end of the copied source file in the following cases:

- When testing static functions
- When using static global variables

Please keep this behavior in mind when preparing and executing tests with TESSY.

Adaptation to target environment

When running tests on a specific target platform, adaptations of compiler options and target debugger settings may be needed within the respective target environment. The verification of the TESSY core workflow covers tests conducted on a Windows host system using the GNU GCC compiler. In order to verify the transmission of test data and expected results to and from the target device, there are tests available that may be executed using the adapted target environment. These tests check the communication layers of the test driver application.

For details on how to run these tests refer to the application note "048 Using Test Driver Communication Tests.pdf" within the TESSY installation directory.

It is recommended to run these tests with your specific compiler/target environment.

Operating limits

TESSY is constructed for usage as a unit testing tool in order to verify the functional correctness of the function under test. The following restrictions and prerequisites for TESSY apply:

- The source code to be tested shall be compilable without errors and warnings by the respective target compiler. TESSY may fail analyzing the interface of the module to be tested, if there are syntactical errors within the source code.
- TESSY does not check any runtime behavior or timing constraints of the function under test.
- The test execution on the target system highly depends on the correct configuration of the target device itself, the correct compiler/linker settings within the TESSY environment and other target device related settings within TESSY (if applicable). Any predefined setup of the TESSY tool for the supported devices requires manual review by the user to ensure proper operation of the unit testing execution.
- The usage of compiler specific keywords and compiler command line settings may require additional tests for tool qualification. Correct operation of the TESSY toolset with respect to the Qualification Test Suite (QTS) test results may only be provided for ANSI compliant C code.

New features in TESSY 3.0

Note the chapter Migrating from TESSY 2.x to 3.x. You will find a list where to find functions that moved within the new version.
Check as well the Practical exercises to learn about TESSY´s new features!

Revised graphical user interface (GUI)

Based on the Eclipse Rich Client platform, the new TESSY GUI provides easier access to all functionalities within multiple perspectives and views on the test data. All TESSY tools are now integrated as individual perspectives within one application (see section 3 Basic handling).

Improved main perspective

The new main perspective Overview provides a more summarized overview about passed and failed results, coverage and other useful status information. The new project structure introduces a folder hierarchy instead of the former single project level. The coverage results are displayed directly within the project structure.

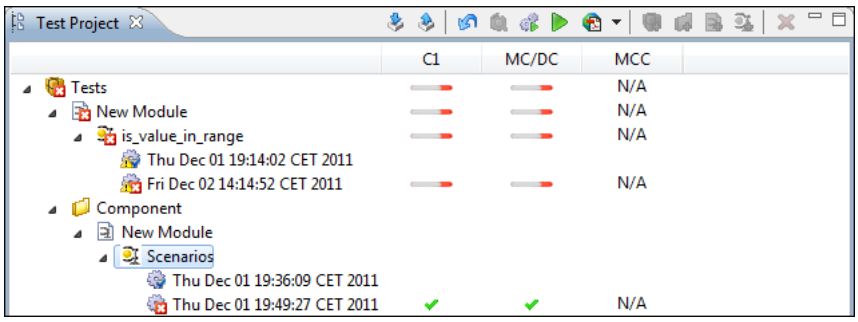


Figure 0.2: New Test Project view

Test results are stored within test runs which allows keeping former (successful) results for reporting while continuing to work on test cases and test data. Grouping of test cases is available for better structuring of test data.

Traceability of requirements

TESSY introduces a basic requirement management solution that allows synchronizing with external requirement sources (e.g. DOORS, Microsoft Word, Microsoft Excel and

other through CVS/XML file import). Reference mechanisms from test cases back to the requirements allow easy coverage analysis and reports as well as impact analysis of requirements changes.

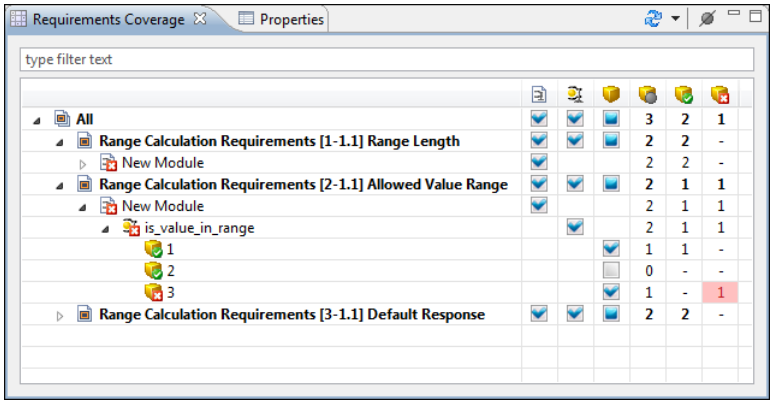


Figure 0.3: New traceability of requirements

Modules, test objects and test cases may be linked to requirements within a convenient requirements coverage view.

Enhanced coverage handling

All available coverage measurements (C1, MC/DC, MCC) may be switched on or off and a minimum required coverage may be defined for individual test objects.

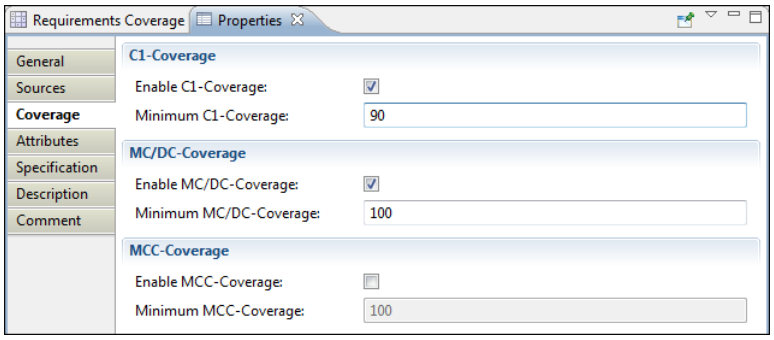


Figure 0.4: Defining coverage measurements

Improved integration of the Classification Tree Editor (CTE)

The CTE has a tighter integration to TESSY and the Test Data Editor (TDE). Values for all types of variables can be specified in the new CTE.

Tabular Test Data Editor (TDE)

The new tabular TDE provides easy overview about test data and entering of test data for multiple test steps at a glance.

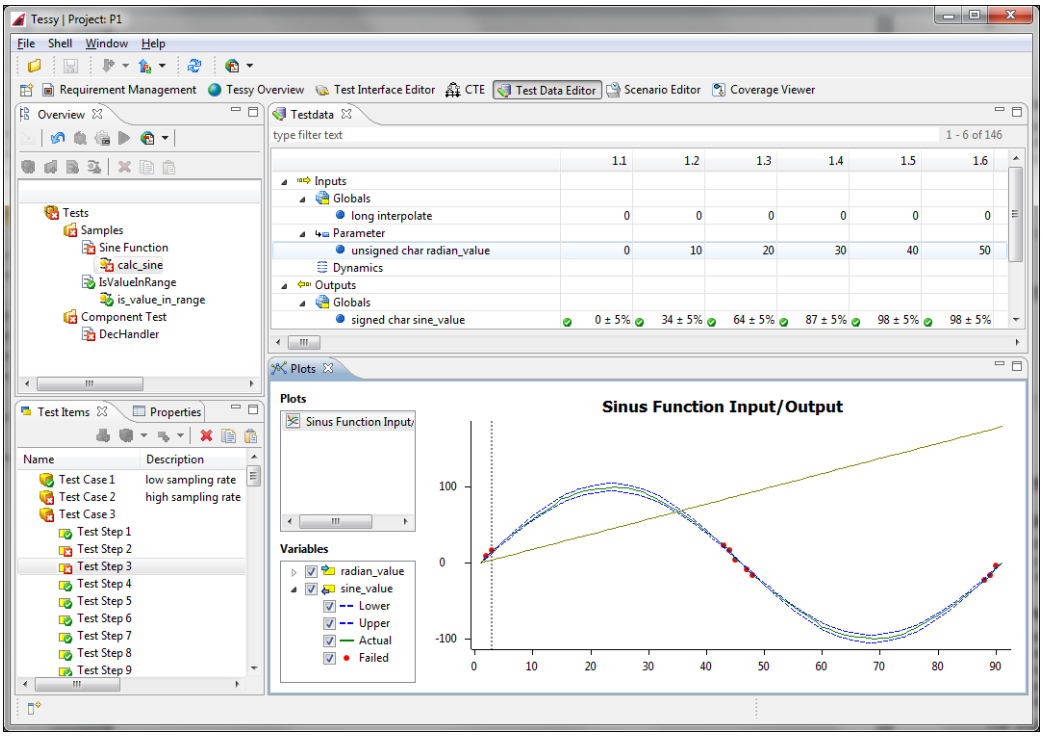


Figure 0.5: New tabular Test Data Editor

Enhanced Usercode editor

The Usercode editor is now an integral part of the TDE perspective and provides flexible editing and outline views together with the tabular test data viewer. Usercode for stub functions may be specified on a per-test-step basis which facilitates returning specific values for individual test steps.. A new evaluation macro results view provides faster access to failed test results.

Improved test data plots

The new plot view provides graphical plots of test data for input and expected values. Plots may be created for an arbitrary number of variables on test case, test step or even on array element level. Each plot is populated with the current test data within a graphical preview. Multiple variables may be added to a single plot.

Enhanced component testing

The component test feature has been improved, so that all variables of the component interface may be set and checked at arbitrary points in time within the scenario. Entering of input data and expected results is carried out using the normal test data view.

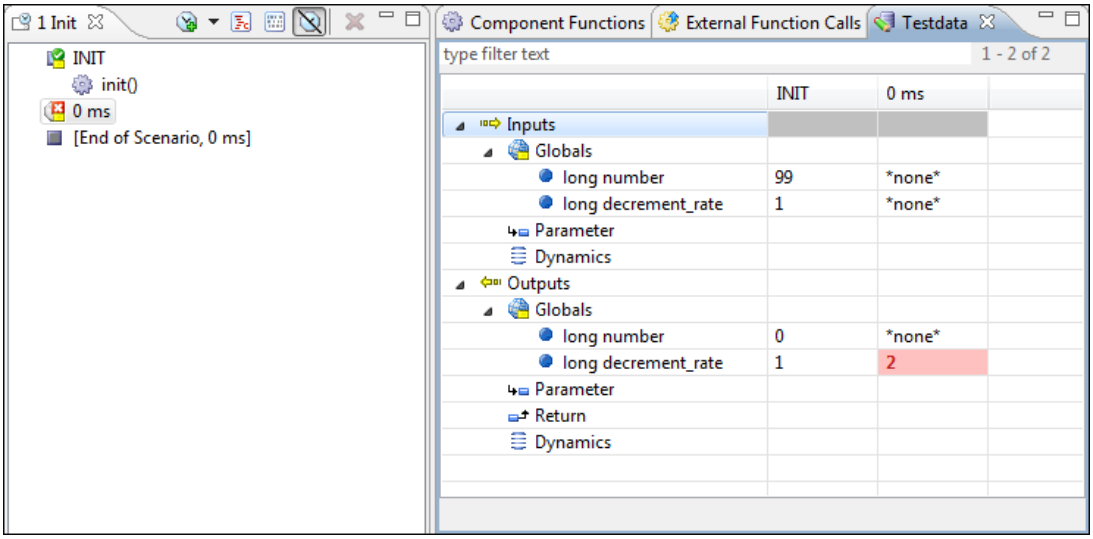


Figure 0.6: New Test Data view

Application Programming Interface (API) and commandline interface

An API for the project database (PDB) and the test database (TDB) is now available, enabling the user to write scripts (e.g. using a DOS shell) to modify these information or add new modules with all necessary settings (e.g. source files, include paths, defines) into the project database. A command line executable provides all commands of the API for usage within DOS batch files.

Troubleshooting

If you have troubles with errors or do not know how to proceed:

- Check this manual and make sure that you have operated correctly.
- Check our application notes that are available in the Help menu of TESSY ("Help" > "Documentation").
- Check our website for commonly asked questions and current issues http://www.razorcat.com.

If you have further questions or if there is an error report, please contact our **Technical Support** per e-mail: support@razorcat.com
Include in your support request e-mail:

- your contact data
- a short description of your question or problem
- the TESSY Support File, if you get errors

The **TESSY Support File** (tgz-file) contains information about test object including data, compiler, project settings etc. It helps the support to detect the cause of your problem.

Creating the TESSY Support File

→ In TESSY, select the module or test object that is causing the problem.

→ Click "Help" in the menu bar.

→ "Support" > "Create Support File"

→ Tick the box "Preprocessed sources", if possible.

Important: The box "Preprocessed Sources" is not ticked by default. This avoids that confidental sources might be included accidentally. Whenever you can afford to provide the sources to the support, tick the box "Preprocessed Sources". In most cases it is necessary for successful problem investigation.

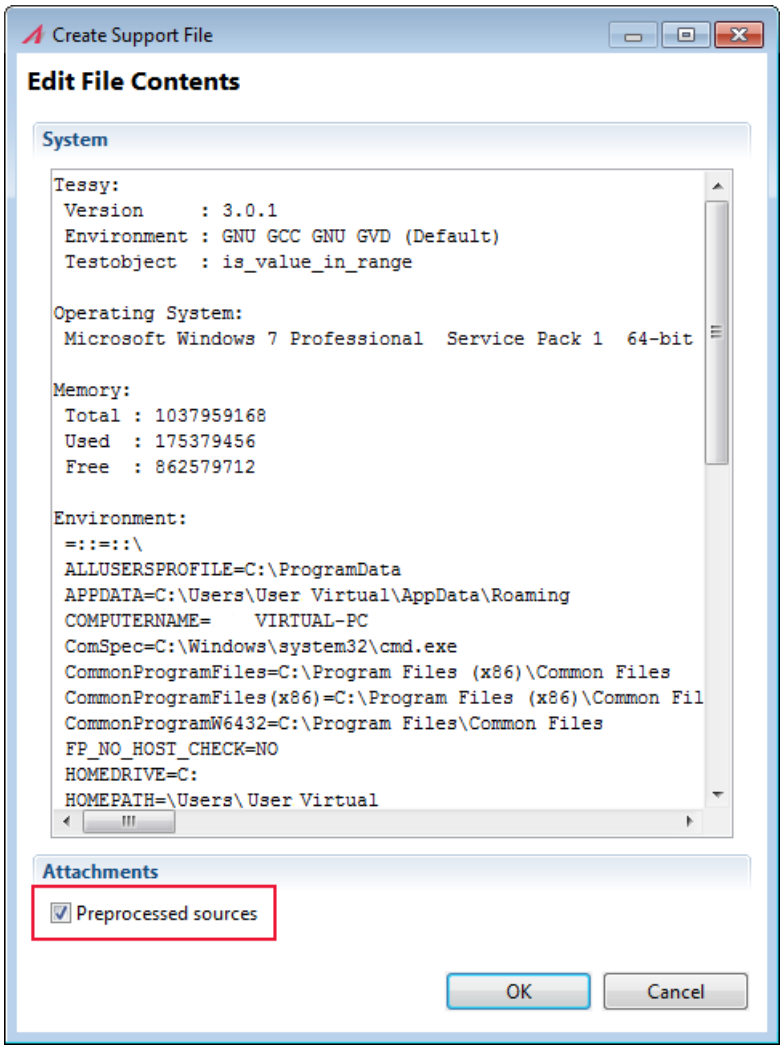


Figure 0.7: Dialog for creating the TESSY Support File

→ Click "OK".

→ Choose a folder and a name for the file and click "Save".

The TESSY Support File (tgz-file) is created.

Tipps for a better TESSY Support File

1. Reduce unnecessary information:

   → Before reproducing the problem, switch to the Console view of the perspective "Overview".

   → In the tool bar click on the icon (Clear Console).

   All messages will be deleted.

2. Enable TESSY to display additional information:

   → In the main menu bar select "Help" > "Logging".

   → Select a process, the level (if selectable) and "on" as the table beneath "Table Name" explains (see table below).

It is possible to enable logging for all process steps.

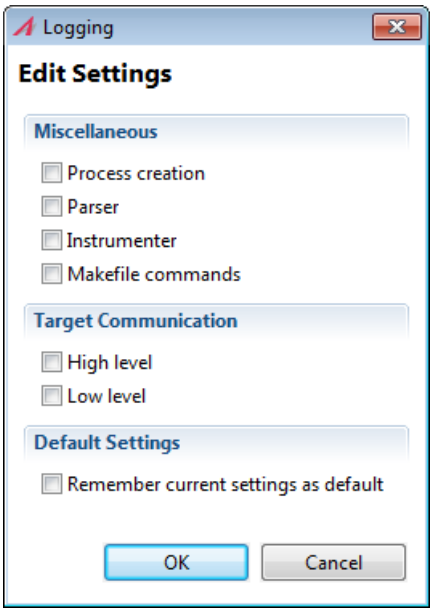


Figure 0.8: Dialog for logging settings

The additional information can relate to different process steps within TESSY. Enable the logging of the information you suspect the problem to stem from:

Process step	**Log if . . .**
Process Creation	parts of TESSY do not start correctly or TESSY is not able to start the test system (e.g. debugger).
Parser	problems occur during analysis of the source code to determine test objects and their interfaces. Often caused by incorrect syntax.
Instrumenter	problems occur during instrumentation of the source code to determine code coverage.
Makefile Commands	the test application (slave) or the test driver (master) cannot be created or are created incorrectly.
Target Communication	problems occur in the communication between TESSY and the debugger (e.g. the test system).
High level	you want to log the general TESSY activities. Seldom required to find a problem.
Low level	you want to log debugger-specific activities. Often very useful.

Table 0.13: Information to log and add to TESSY Support File

→ You can save the settings for logging with ticking the box "Remember current settings . . . ".

3. Reproduce the problem:

→ Do the actions again that lead to the problem (e.g. opening the module).

→ Keep the respective element selected that caused the problem (e.g. the test object in case of errors while executing) when creating the support file.

4. Create the support file, now including all log messages.

1 Installation and registration

1.1 Technical requirements

- Windows 8 (32bit/64bit), Windows 7 (32bit/64bit), Windows Vista (32bit/64bit), Windows XP (ServicePack 2 or higher) or Windows 2000 (ServicePack 4 or higher).
- TESSY 3.0 can be installed and run in parallel to any previous TESSY version.
- For the reports you need at least Adobe Reader 7.0.
- To run TESSY 3.0 you need at least a 1.5 GHz CPU and 2 GB RAM for TESSY.

1.2 Setup

Warning: Deactivate your firewall or virus scanner temporarily while installing TESSY! Otherwise the firewall can cause problems during the installation. Note that some firewalls and anti-virus software can limit the functionality of applications, including TESSY and might need to be modified. For further help ask the producer of the firewall or anti-virus software.

Important: You need local administrator privileges to install TESSY!

The TESSY version 2.9 and later allows you to have multiple TESSY installations with different versions on the same computer. You do not have to uninstall an older version.

To install TESSY 3.0 on your computer,

→ insert the TESSY-DVD into the DVD drive and wait for the setup program to start OR start the installation program manually ("tessy[version number].exe").

The InstallAware Wizard will start. This will take a few moments.

→ When the InstallAware Wizard is ready, click "Next".

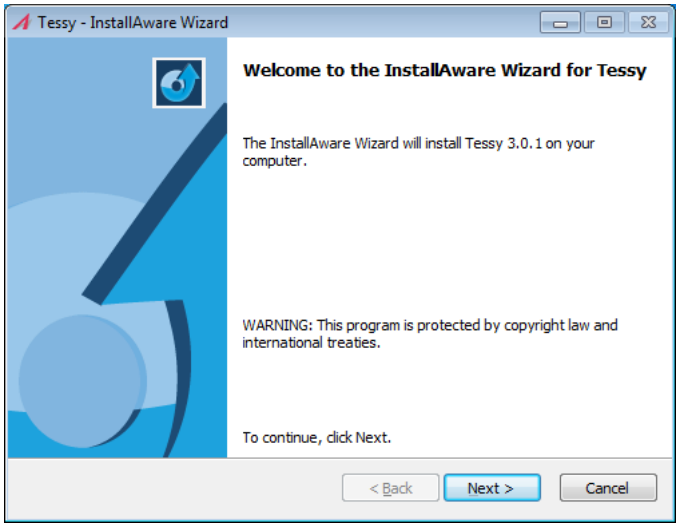

Tessy - InstallAware Wizard
Welcome to the InstallAware Wizard for Tessy
The InstallAware Wizard will install Tessy 3.0.1 on your computer.
WARNING: This program is protected by copyright law and international treaties.
To continue, click Next.
< Back
Next >
Cancel

Figure 1.1: InstallAware Wizard

→ Read the license agreement carefully. Check the box to accept and click "Next".

Tessy - InstallAware Wizard
License Agreement
Please carefully read the following license agreement.
Tessy V3.0
(c) 1998-2011, Razorcat Development GmbH and Daimler AG
License Agreement
Razorcat Development GmbH (in the following named Razorcat), provides the guarantee, and the licensee, according to the following terms, accepts a non-exclusive licence or sublicence to utilize and reproduce software in binary format, developed by Razorcat or a third party and distributed by Razorcat or a third party.
Conditions of Use
I accept the terms of the license agreement
InstallAware
< Back
Next >
Cancel

Figure 1.2: License Agreement

→ Now select the setup type: "Complete" (default) is recommended. click "Next".

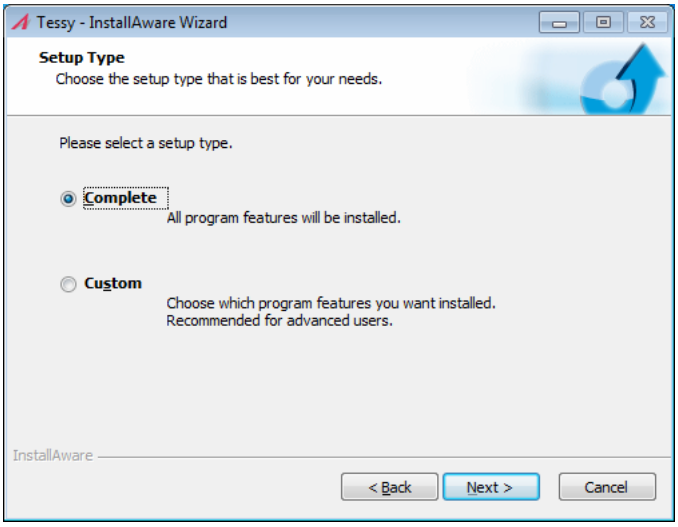


Figure 1.3: Setup Type

→ Select the destination folder (default "`C:\Program Files\Razorcat`") and click "Next".

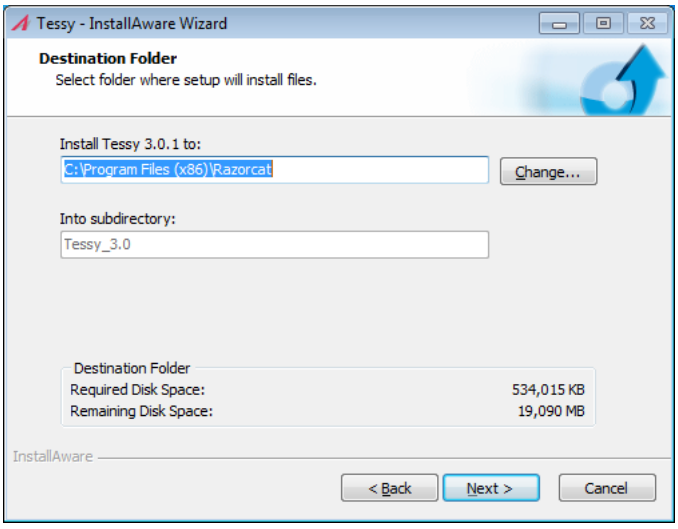


Figure 1.4: Destination Folder

→ Select the TESSY testarea folder ("Folder for temporary files:"; default "`C:\tessy`") click "Next".

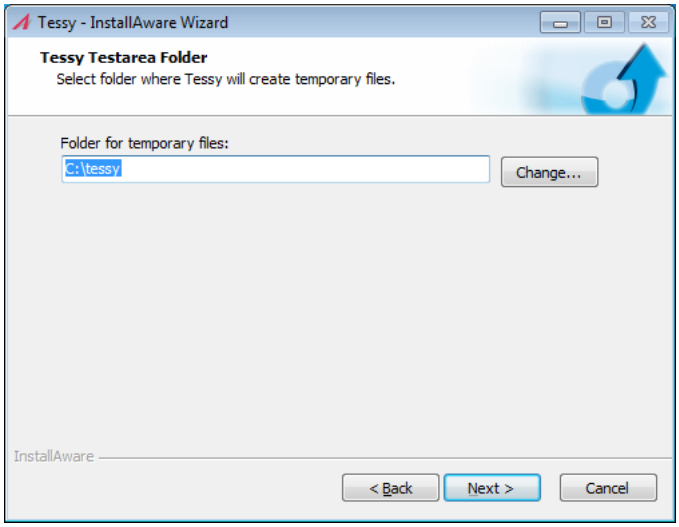


Figure 1.5: TESSY Testarea Folder

→ Select the program folder (default "TESSY 3.0") and decide, whether the installation on the system is for "all users" or for a certain user. Click "Next".

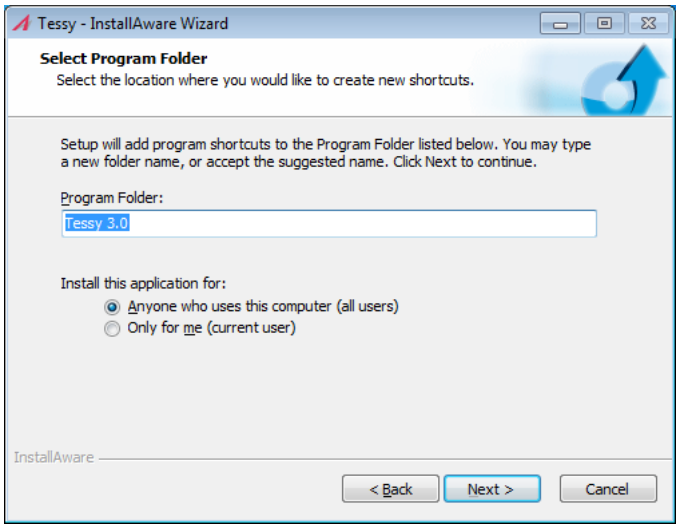

Tessy - InstallAware Wizard
Select Program Folder
Select the location where you would like to create new shortcuts.
Setup will add program shortcuts to the Program Folder listed below. You may type a new folder name, or accept the suggested name. Click Next to continue.
Program Folder:
Tessy 3.0
Install this application for:
Anyone who uses this computer (all users)
Only for me (current user)
InstallAware
< Back
Next >
Cancel

Figure 1.6: Program folder and user selection

→ Start the installation by clicking “Next”.

The installation will take a few moments.

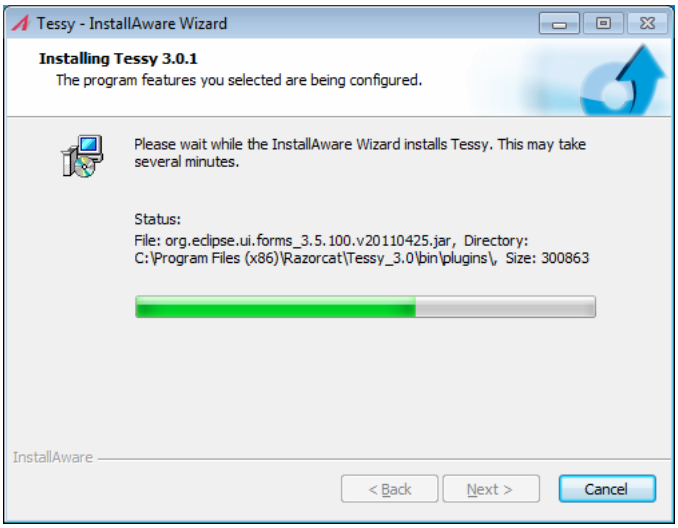

Tessy - InstallAware Wizard
Installing Tessy 3.0.1
The program features you selected are being configured.
Please wait while the InstallAware Wizard installs Tessy. This may take several minutes.
Status:
File: org.eclipse.ui.forms_3.5.100.v20110425.jar, Directory: C:\Program Files (x86)\Razorcat\Tessy_3.0\bin\plugins\, Size: 300863
InstallAware
< Back
Next >
Cancel

Figure 1.7: Installing process of TESSY

→ When the InstallAware Wizard is completed, click “Finish”.

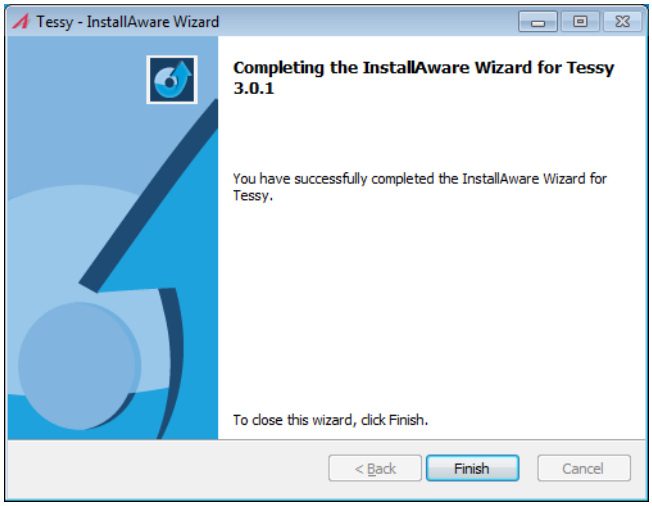

Tessy - InstallAware Wizard
Completing the InstallAware Wizard for Tessy 3.0.1
You have successfully completed the InstallAware Wizard for Tessy.
To close this wizard, click Finish.
< Back
Finish
Cancel

Figure 1.8: Installation is completed

Now you have completed the installation of TESSY.

1.3 Registration

→ In Windows select "Start" > "All Programs" > "TESSY 3.0".

The License Manager will start with a License Key Request popup window.

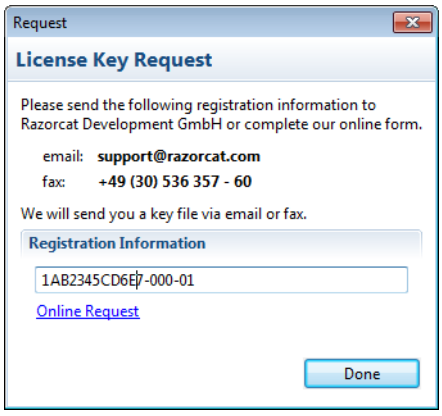

Request
License Key Request
Please send the following registration information to Razorcat Development GmbH or complete our online form.
email: support@razorcat.com
fax: +49 (30) 536 357 - 60
We will send you a key file via email or fax.
Registration Information
1AB2345CD6E7-000-01
Online Request
Done

Figure 1.9: License Key Request

→ If you do not have a license key: Click on "Online Request" and fill out the form for an evaluation key request. In return you will get a license key via e-mail within a license key file. The license key file is a plain text file with the ending .txt.

Important: The license key file is not generated and send out automatically, therefore it can take some time until you receive it.

→ If you have already received your licence key in a license key file: Click on "Done".

The Configure window for the License Server will open (see figure 1.10).

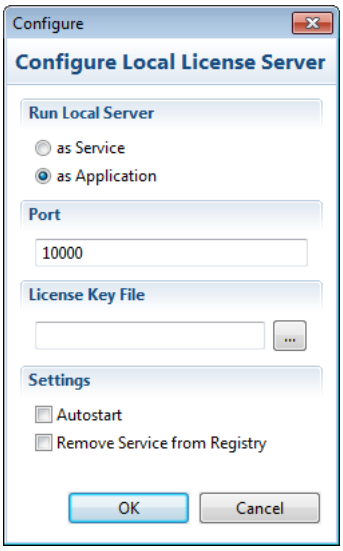


Figure 1.10: Configure menu of the license server

→ Under "License Key File" choose your license key file (*.txt).

→ Tick the box "Autostart" to start the license server hereafter in the background. The system will try to set the autostart. If you do not have administrator privileges, the autostart is set for your windows user.

→ Optional: You can run the license server as service (select "Service" under "Run Local Server"). The default is "Application". If you wish to remove the service from the registry, tick the accordingly box under "Settings".

Important: You need administrator privileges to run the license server as service and to remove it from the registry!

→ Click "OK".

The license server will start automatically. If not, click on (Start Local Server).

→ Close the License Manager and start TESSY by selecting "Start" > "All Programs" > "TESSY 3.0".

1.3.1 The License Manager

→ To start the License Manager manually select "Start" > "All Programs" > "TESSY 3.0" > "Floating License Manager".

→ To start the server manually click on the icon (Start Local Server).

→ To stop the server manually click on (Stop Local Server).

→ To configure the server click on (Configure).

→ To check your license key file click on (Check). This can be useful to determine the error if the server does not start.

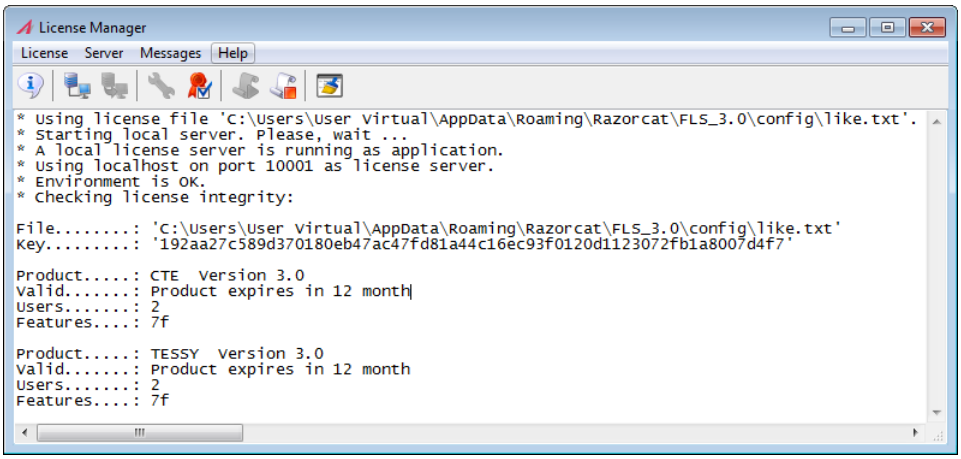

License Manager
License Server Messages Help
* Using license file 'C:\Users\User Virtual\AppData\Roaming\Razorcat\FLS_3.0\config\like.txt'.
* Starting local server. Please, wait ...
* A local license server is running as application.
* Using localhost on port 10001 as license server.
* Environment is OK.
* Checking license integrity:
File.....: 'C:\Users\User Virtual\AppData\Roaming\Razorcat\FLS_3.0\config\like.txt'
Key.....: '192aa27c589d370180eb47ac47fd81a44c16ec93f0120d1123072fb1a8007d4f7'
Product.....: CTE Version 3.0
Valid.....: Product expires in 12 month
Users.....: 2
Features....: 7f
Product.....: TESSY Version 3.0
Valid.....: Product expires in 12 month
Users.....: 2
Features....: 7f

Figure 1.11: License key check successful: this license key is correct

1.3.2 Solving problems

If the license server does not start, or you get any errors e.g. "a license server is not running", check your license key:

→ Start the License Manager manually, if it has not startet yet: "Start" > "All Programs" > "TESSY 3.0" > "Floating License Manager".

→ Click on (Check) to check your license key file.

→ Check the error message (see figure 1.12).

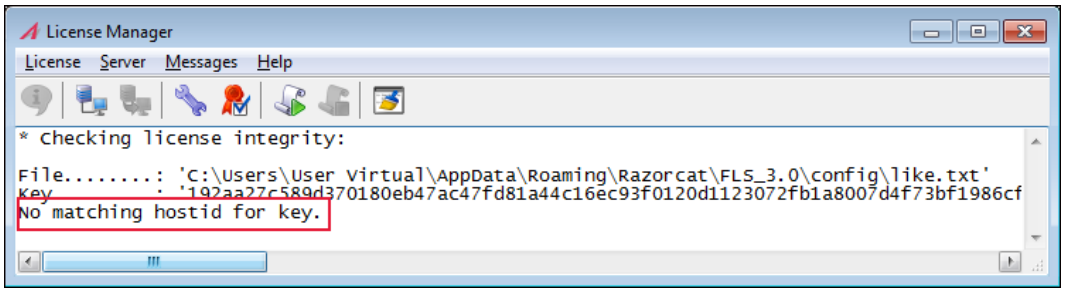


Figure 1.12: License key check unsuccessful: license key is incorrect for the host id

In many cases you can already determine the problem with the help of the error message. In case of the error "No maching hostid for key" the license key does not match to the host id of your computer:

→ Configure the correct license key file in the manager: Click on (Configure) and select the correct license key file. Click "OK".

→ Click on to check the license key file again.

→ If the error still appears, contact our support (see Troubleshooting) to get a new license key file.

1.4 Uninstallation

Important: By uninstalling TESSY the project root will not be deleted, neither your project data or your configuration file will be deleted. But anyway make sure that your data is saved.

To uninstall and remove all components of TESSY,

→ in Windows click "Start" > "All Programs".

→ "TESSY 3.0" > "Uninstall TESSY"

All components of TESSY will be removed. This will take a few seconds.

2 Migrating from TESSY 2.x to 3.x

In the new version of TESSY you will find many new functions as mentioned in section New features in TESSY 3.0. Therefore, the user interface has changed in some ways and known features moved.

2.1 Changes in handling

To make it easy to switch from working with TESSY 2.9 to TESSY 3.0, the following table shows and explains functions and new handling:

Theme	**Handling in TESSY 2.9**	**New handling in TESSY 3.0**
analyzing modules	A double click on a module starts the interface analysis and the module is opened. In case of interface changes, a reuse is required before continuing working with the module.	Start the analysis by clicking on the icon of the tool bar or on the plus sign of the module tree node. You can as well use the context menu. If there are any test objects ready to be tested, the module can be opened. On interface changes, a reuse can be carried out for individual test objects. Other test objects that are still up to date can be executed independently. New test readiness status icons provide a quick overview about the test object status.
coverage measure-ment	Results of the coverage measurements are shown in addition to the results of the test execution.	Coverage measurement results are part of the overall test result for a test object. The available coverage measurements (i.e. C1, MC/DC, MCC) can be enabled or disabled. Threshold values for successful coverage can be defined.
CTE	Assigning variables to classifications and classes in CTE is restricted to certain variable types.	All variables of the test object interface can be assigned using CTE. The full featured test data view is available for entering data within the CTE perspective of TESSY.

continue next page

Theme	Handling in TESSY 2.9	New Handling in TESSY 3.0
IDA (reuse of test data)	All test objects of a module need to be assigned to the new interface for a reuse operation.	The new IDA allows reusing individual test objects while leaving the other ones untouched (they may be reused later). All reused test objects are ready for execution independently of the status of other test objects within the same module.
module properties	The module properties dialog provides the means to change any module settings.	Module settings are specified on all levels of parent elements of modules within a test project (e.g. test collection and folder). These settings are inherited by modules and can be overwritten on module level.
PDB file	Within the PDB file all tests for a specific software project are stored. It is usually stored beneath the project root of the project. It contains a list of "projects". Each project has a list of modules as children.	All tests for a specific software project are stored now within a database that is located beneath the project root directory of this software project. This is called a "test project" which contains a list of "test collections" that can be further structured using "folder" trees. Each folder contains "modules".
reports	Reports are generated in HTML, Word and Excel format using python scripts.	Reports are available in PDF format only.
Command line tool	The "tessycmd" executable provides basic operations to import and execute tests using the command line.	The "tessycmd" executable provides full control for the overview perspective in TESSY. This allows creating whole test projects and executing tests as well as importing and exporting of test data.
result plots	Result plots of output variables are based on the assignment of a special evaluation mode.	Result plots are defined independently from the evaluation mode for any number of variables. Each result plot can include mixed input and output variables.
TDE	The test data editor TDE shows the test data of one test step.	The new tabular TDE shows multiple test steps and test cases as a spread sheet. Input values and expected values including the evaluation mode are entered using an inline editor.

continue next page

Theme	Handling in TESSY 2.9	New Handling in TESSY 3.0
test execution	The test cases to be executed need to be selected prior to execution.	Selecting the test object within the test project view allows execution of all test cases without further test case selection. The test item view provides selection and execution of individual test cases.
test results	Test results are available after execution of the test and any changes of test data void the results.	The test item view displays the current execution results, if available. All previous test results are also available within the separate test results view which provides a history of the latest test runs. Any historical test run can be selected to provide the current result (e.g. for reporting).
TIE	The interface editor TIE shows the interface of one test object.	The interface can be displayed on module level showing all defined struct, union and enum types as well as on test object level showing the interface specific for this test object.
usercode	The user code editor UCE shows the user code for a test object and is opened as separate tool beside the TDE	The TDE perspective within TESSY includes editors for all user code parts and an outline view showing all available user code fragments.

Table 2.1: Changes in handling with TESSY 2.9 and TESSY 3.0

2.2 Importing previous projects

Warning: Proceed as described below, otherwise there might be some damages to your data!

Important: With TESSY versions 2.6 and previous you need TESSY 2.9 to convert your databases to the new format. Download TESSY 2.9 from http://www.razorcat.com. Contact our product support if assistance is needed.

Version	HowTo
TESSY 2.9	Importing test data requires the following steps: • Create a new project in TESSY 3.0 to be used for the import. Use the default configuration. • Open the Test Environment Editor (TEE): "File" > "Edit Environment". • Activate your compiler and debugger. • Next to the tool bar click on "File" and choose "Show all". • Drag and drop the configuration from "Configuration" > "System" to "File" beneath. By default, the GCC environment is already used in "File". Configuration files from Tessy 2.9 cannot be used in TESSY 3.0! You have to merge all settings into the new configuration file of TESSY 3.0. Not all settings can be taken into the new configuration file: The Make Call has been changed in TESSY 3.0. All attributes which are needed within the makefile template are automatically generated by TESSY. For this purpose the attributes in question gets the special flag "Makefile Variable". For instance, the attribute "Linker File" will be generated in the test object makefile as "***LINKER_FILE***". The makefile templates of Tessy 2.9 are not usable in TESSY 3.0: You have to merge your settings into the new makefile template of TESSY 3.0. The TESSY 3.0 makefile templates have the version number of 2 to distinguish the new makefile templates from former versions.
TESSY 2.6	You have to convert your projects to version TESSY 2.9 first! • Open your projects in TESSY 2.9. The TESSY converter will ask you to convert the databases to the new format (*.pdb). • Continue as described above with TESSY 2.9.

continue next page

Version	HowTo
TESSY 2.5	You have to convert your projects to version TESSY 2.6 first! • Create a module backup file ("Module" > "Save") for each module which you want to use. To restore all tmb files within their project folders, please name the tmb file as follows: [project name].[module name].tmb • Save all tmb files in a folder of your choice, e.g. `tmb2.5_[project database name]`. • Open TESSY 2.6 and create a new project database. Adjust the project root, configuration file, etc., so that the project settings are identical as in TESSY 2.5. • Select the backup folder using "File" > "Database Backup" > "Backup Directory", e.g. `tmb2.5_[project database name]`. • Restore all backup files using "File" > "Database Backup" > "Select". • In the restore from backup dialog, select all tmb files. TESSY 2.6 will now restore and to convert the tmb files. In addition TESSY creates the respective project folder for the modules to be imported. • Continue as described above with TESSY 2.6.
TESSY 2.4, previous	• Convert the projects to the next major version of TESSY up to TESSY 2.5 (e.g. TESSY 2.4 > TESSY 2.5 > TESSY 2.6). Please contact our product support if assistance is needed. • Continue as described above with TESSY 2.6.

Table 2.2: How to import previous projects in TESSY 3.0

2.3 Missing features

Some features are obsolete while others are not yet available within this release of TESSY 3.0.

2.3.1 Python based report

All TESSY 3.0 reports are created as PDF files, the former Python based report generation scripts are not available any more. Reporting is now based on the Eclipse BIRT

reporting framework. Report adaptations can be carried out by changing the BIRT report templates. For more information refer to the application note "008 Customizing Reports" in the Help menu of TESSY ("Help" > "Documentation").

2.3.2 THAI

The TESSY hardware abstraction interface is not yet available.

2.3.3 OBT

The original binary test feature is not yet available.

2.3.4 Alias names

The display of alias names for interface variables is not yet available.

2.3.5 Debugger Scripting Prolog / Epilog

The support of usercode script code as prolog / epilog for TRACE32 and HiTOP debuggers is not yet available.

2.3.6 Generation of test steps

This feature is not available within TESSY 3.0. It will be replaced by a more powerful implementation and it is planned to become available within TESSY 3.1.

2.3.7 Diff Viewer

The "Diff Viewer" application of TESSY 2.9 which showed the differences between two result XML files is not available within TESSY 3.0. This feature will be replaced by a more powerful implementation and will become available within a later version of TESSY.

3 Basic handling

This chapter explains how to create databases for your test, how to work with the different files and the graphical user interface of TESSY and provides some information about useful shortcuts to work more efficient.

3.1 Creating databases and working with the file system 18
3.1.1 Creating a project database 18
3.1.2 Optional: Using a specific environment setting 22
3.1.3 Deleting a project 23
3.2 Understanding the graphical user interface 25
3.2.1 Menu bar 25
3.2.2 Tool bar 26
3.2.3 Perspectives and perspective (tool) bar 26
3.2.4 Views 27
3.2.5 Status bar 30
3.3 Using the context menu and shortcuts 31
3.3.1 Context menu 31
3.3.2 Shortcuts 31

3.1 Creating databases and working with the file system

The following table provides a fast overview about TESSY´s file system and databases:

What	does
configuration file	Contains appropriate target environments for the project database which can be defined with the Test Environment Editor and will be stored under `[project root]/TESSY/config`.
persist (folder)	Contains the databases for the project, one for requirements and test collections, the other one for test data.
project database	Specifies the root directory of one of your projects. You should have one TESSY project for each of your development projects.
project root	Specifies a root directory of your projects, so that all paths (e.g. sources, includes, etc.) can be related to this root. Every project will have an own project root.
tmb-files	Format to store your backups, usually under `[project root]/tessy/backup`.
work (folder)	Contains all temporary files generated during the test process. This entire directory can be deleted without loosing any data of the TESSY project.
workspace	Contains all Eclipse-related settings for TESSY (layout/size of windows/perspectives/views) and the list of projects (the file projects.xml). You can close TESSY, move the projects.xml somewhere else, delete the entire directory, restart TESSY and restore the projects.xml.

Table 3.1: File system and databases of TESSY

3.1.1 Creating a project database

→ Start TESSY by selecting “All Programs” > “TESSY 3.0” > “TESSY 3.0”.

Loading TESSY will take a few seconds.

Figure 3.1: Loading TESSY

The Select Project dialog will open. At top you can see the path of your workspace (see figure 3.2).

If you have opened a project before, TESSY will open the last project database. To create a new project, select "File" > "Select Project" in the menu bar.

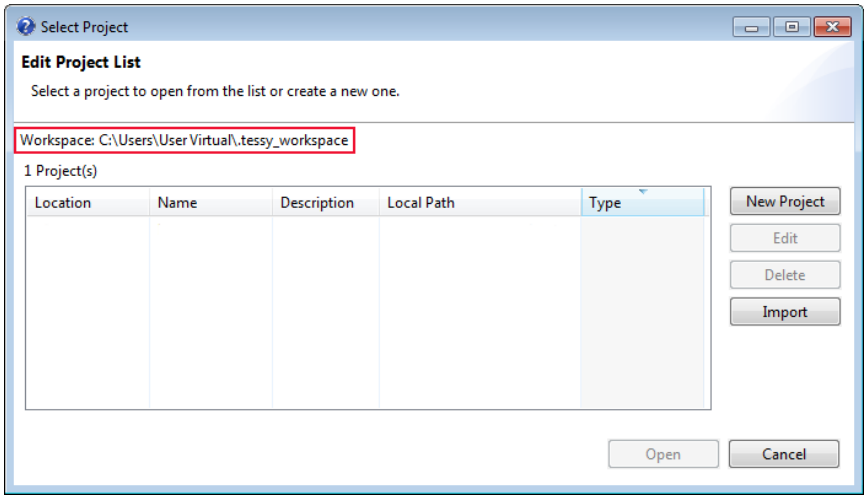


Figure 3.2: Path of the workspace

→ Click "New Project".

→ Enter the name of your project, e.g. ***Example1***.

Important: Do not use special characters for your projectname, e.g. /"!*[]<> etc. Only simple brackets, ( ), dash - and underscore _ are permitted.

→ Optional: Enter a description of your project.

→ Select a project root: Click on "...", create a new folder where your project files are supposed to be located, e.g. "`MyProjects\Example1`" and select the folder as project root (see figure 3.3).

All the project-related TESSY databases containing information on the test environment, referenced source files, compiler, debugger, etc. will be stored within a sub folder of the project root and all paths into your project will relate to this root.

Creating a new project

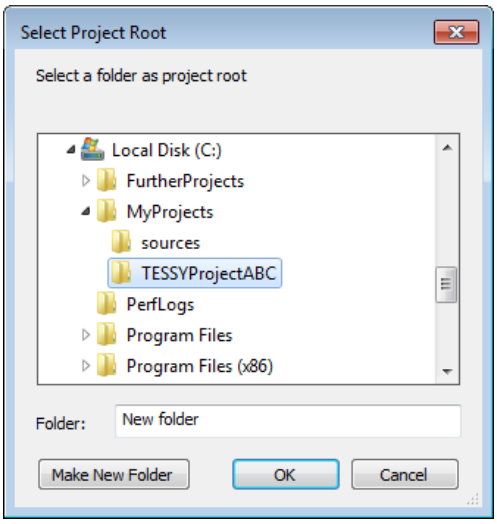


Figure 3.3: Selecting a folder for the project root.

→ Click "OK".

TESSY will automatically propose the location of the configuration file containing the default environment (GNU gcc). You may later open the TEE to enable other available compiler/target environments.

The "Database Root Path" will be defined automatically (see figure 3.4).

→ Click "OK".

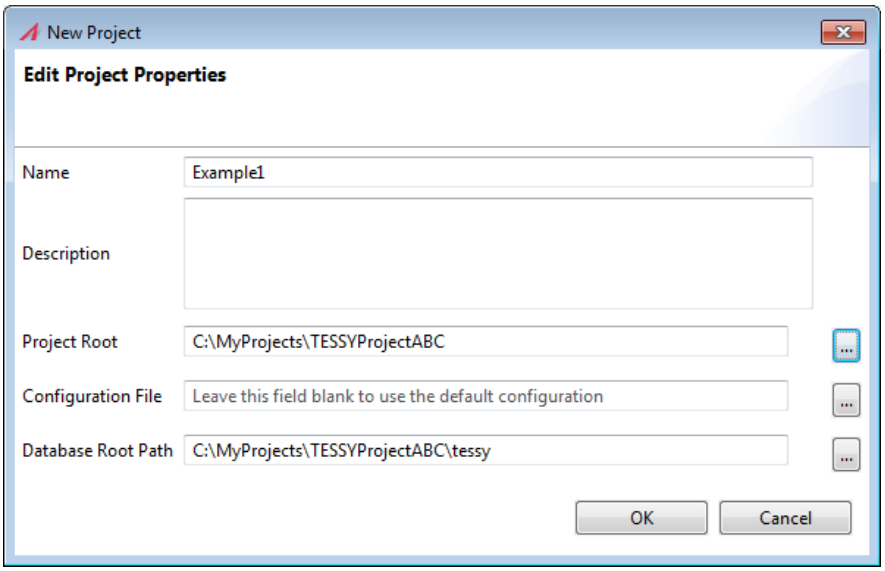


Figure 3.4: Creating a new project

Now TESSY creates automatically a sub folder "tessy" within the project root directory. This folder contains (within sub folders) the configuration file and the persistence databases. This will take a few seconds.

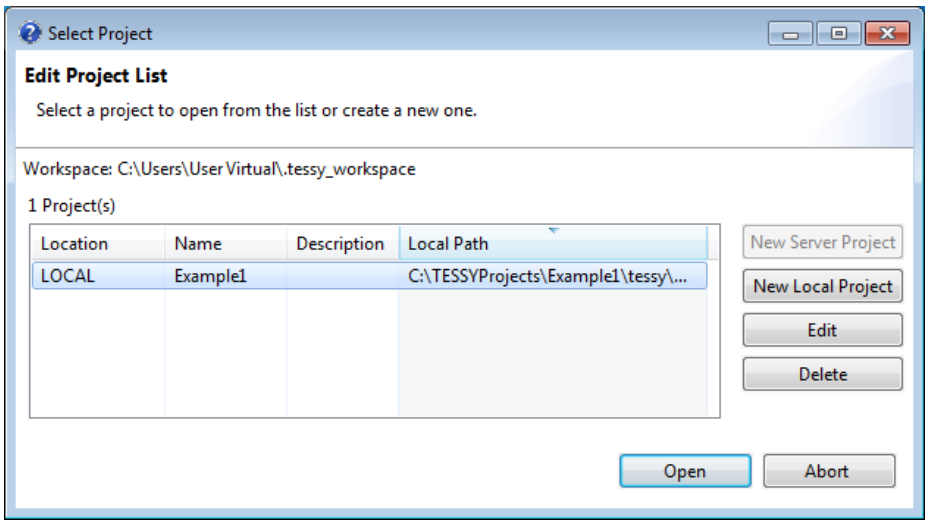


Figure 3.5: Project Example1 is created.

→ When TESSY has finished creating the project, select the project name and click "Open".

TESSY will start and open the Overview perspective. The view "Test Project" is still empty.

The project root

The project root contains your development project, e.g. source files, and one sub folder "tessy" that contains all TESSY related files.

With the project root set you will be able to use relative paths within the TESSY configuration, e.g. references to source and config files. This ensures that you can move whole projects to new locations.

You can move your whole project directory and then change the project root:

→ "Window" > "Preferences"

→ Project Attributes: TESSY_PROJECT_ROOT

Make sure that you also move the project directory in this case so that the paths still match. Otherwise you will get errors when opening the project again!

3.1.2 Optional: Using a specific environment setting

Important: This section is only recommended for advanced users that have already worked with TEE. For basic handling we recommend to continue with section 3.2 Understanding the graphical user interface and following sections and then return to this section.

TESSY will create a specific configuration file for each project database. This way you can share the environment settings with other members of your development team. The configuration file should be stored within your project root together with other project related files (e.g. source files). Such a configuration file may contain only the compiler/target environments you want to use. All other environment configurations are not visible for the user as long as this file is assigned to a given project database.

Creating a configuration file

You need administrator privileges to change the default configurations (except User File section). If you do not have enough privileges, TEE will inform you and save your changes in a user specific file.

To create, change and save a new custom configuration file:

→ Open the Test Environment Editor TEE ("File" > "Edit Environment").

→ Select "File" > "Configuration File" > "New. . ."

TEE will insert a new file node within the configurations section.

→ Copy an available configuration environment from paragraph "Configurations" > "System" onto the file node by simply using drag and drop.

→ Now change the settings to your needs, e.g. delete some debuggers from the configuration if they are not necessary for your test environment.

Important: Change the settings in paragraph "file". Only these changes are transferred to the configuration file!

→ Save the changes with "File" > "Save".

TEE will open the save configuration file dialog.

You can now assign the configuration file to a project database of TESSY.

3.1.3 Deleting a project

Warning: Deleted files cannot be restored! Before deleting make sure the project is not needed anymore or backup the project as described in section 6.11 Backup, restore, version control.

→ Close the project ("File" > "Close Project").

→ Open TESSY. The "Select Project" dialog will open automatically.

→ Select the project and click on "Delete" (see figure 3.6).

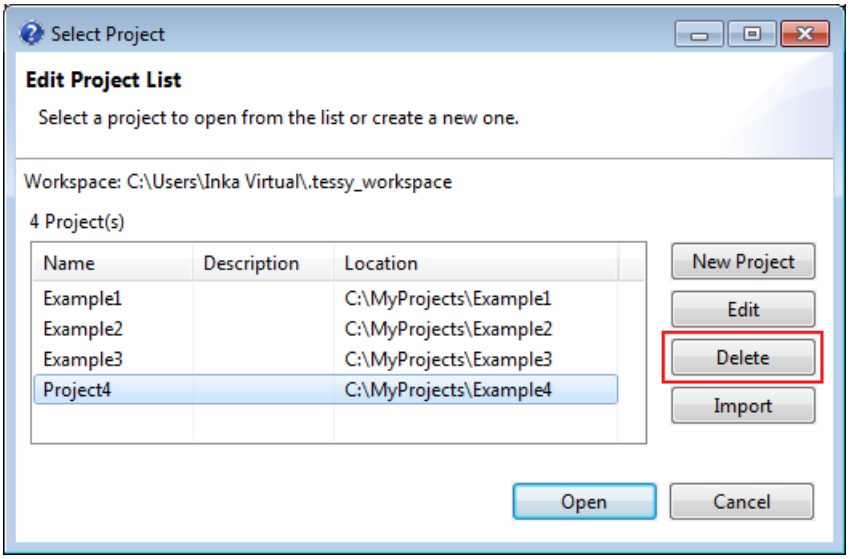


Figure 3.6: Deleting a project

Important: A project will be replaced when a new project with the same name is opened or imported! In the Select Project dialog every project needs an unique name. The replaced projects will not be deleted: You can restore the replaced projects by clicking on the Import-button in the Select Project dialog. In that case the project with again replace a project with the same name.

3.2 Understanding the graphical user interface

When TESSY starts the first time, the graphical user interface (GUI) will open within the Overview perspective.
Please check the terminology shown in the figure "TESSY interface" and the explanations beneath. This terminology will be used throughout this manual.

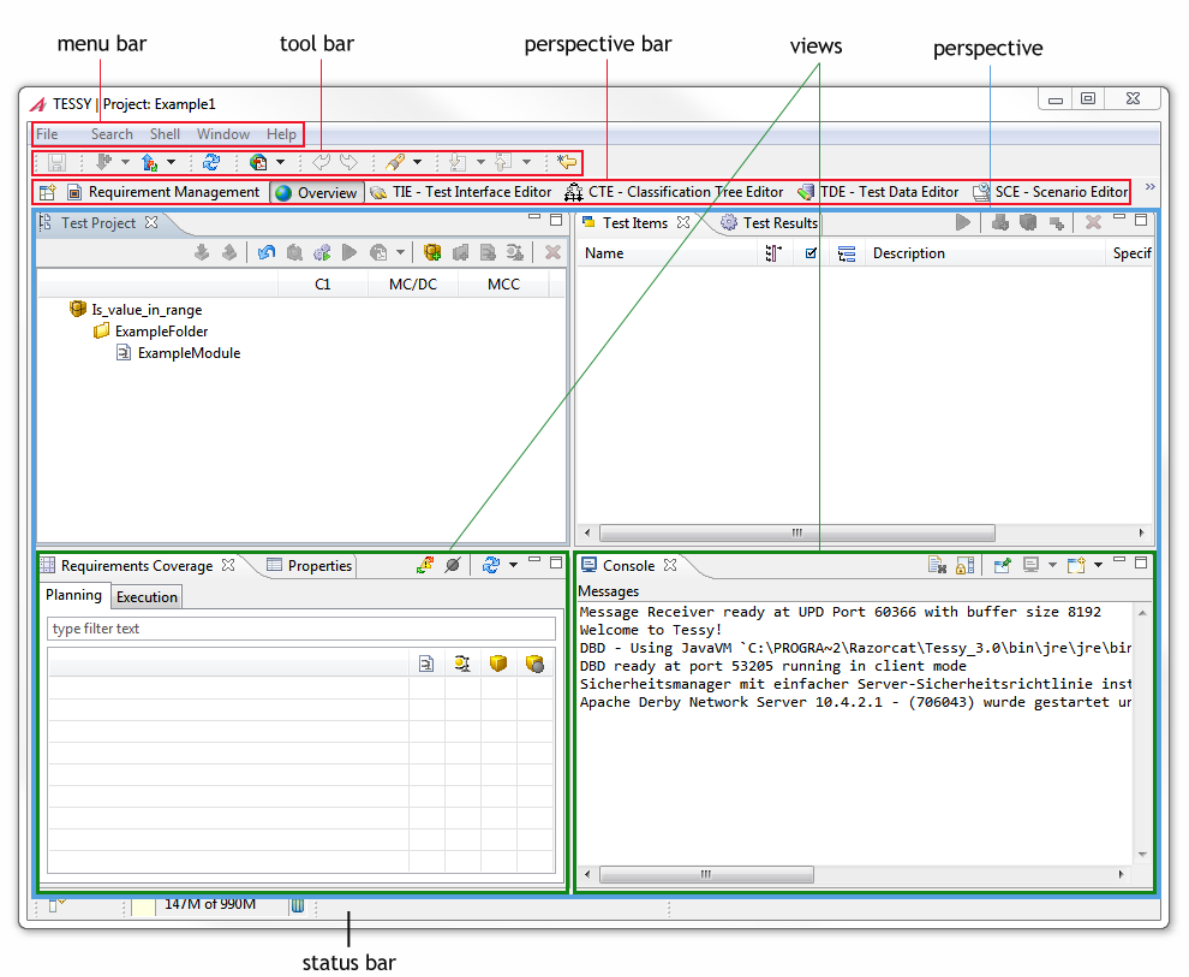
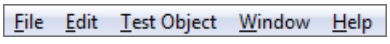


Figure 3.7: TESSY interface and its terminology

3.2.1 Menu bar



The menu bar provides actions as "File", "Windows" etc. Within the "Help" you find the online help of TESSY. Many of these actions may also be available within the context

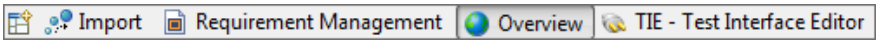
menu of individual views, if the actions are applicable for the items within the view.

3.2.2 Tool bar

At the global tool bar of TESSY interface you can select a project, save changes etc. By rolling over an icon with the cursor a tooltip will tell you the purpose of each icon. There may also be individual tool bars within the views. Here you find the tools for operating, e.g. to start the test execution .

Save all changes in any views or editors by clicking the save icon .

3.2.3 Perspectives and perspective (tool) bar



TESSY contains several perspectives to present information based on different tasks in the test workflow ("Requirement Management", "Overview", "TIE" etc.). Each perspective offers several views. In the perspective bar (containing the perspective names) you can switch between the different perspectives. The perspectives - from the left to the right - follow the actions taken while preparing, running and evaluating a test.
Every perspective name has several right click menu options (the context menu).
By clicking on the left symbol you can open other perspectives (see figure 3.8):

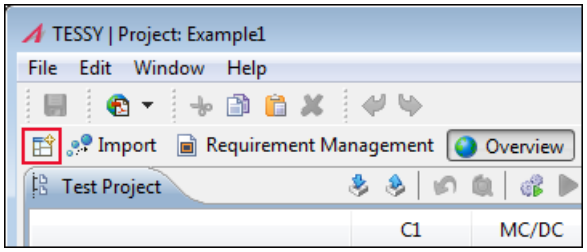


Figure 3.8: Open other perspectives

3.2.4 Views

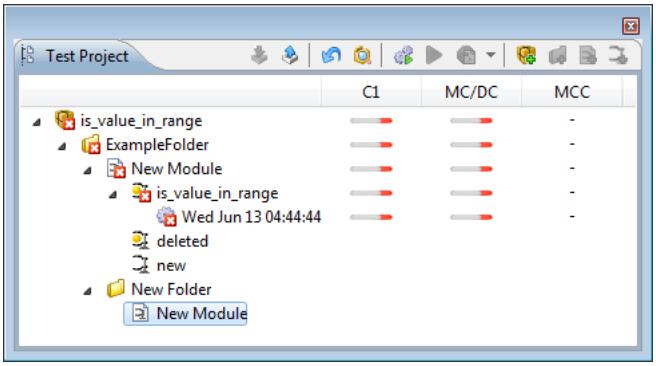


Figure 3.9: Test Project view within the Overview perspective

Every view in each perspective contains possibilities of configurations or provides presentations of data. Some views are common to several perspectives (such as the properties of sources), and some are specific to one perspective (such as the plots in the TIE).

Notice that the views appear differently combined with other views, e.g. the view Test Results within the Overview perspective is combined with the Test Items view, but within the TDE perspective it is combined with the Test Project view. The reason for the different combinations is to give you a fast overview and comparison between various information within each project step.

Adding views to a perspective

You can change the appearance of views for you own needs and open views of one perspective into another perspective:

→ Activate (open) the perspective where you want to add a view.

→ Click "Window" > "Show View".

A window displaying all views will open (see figure 3.10).

→ Select the view you want to open.

→ Click "OK".

The selected view is added to the active perspective.

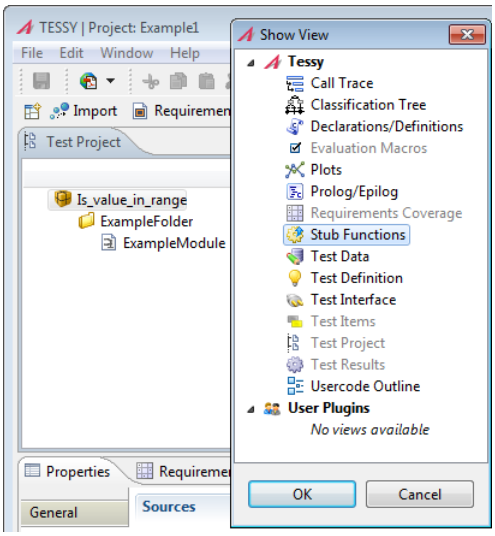


Figure 3.10: Adding views to a perspective

Changing view position

Change the position of views with drag and drop:

→ Click on a name of a view and move it where you like: You can move views to another location within the same group of views or into another group of views or even outside the current window.

→ Right-click on the perspective switch and choose "Reset" to switch back to the original positions of all views of the respective perspective (see figure 3.11).

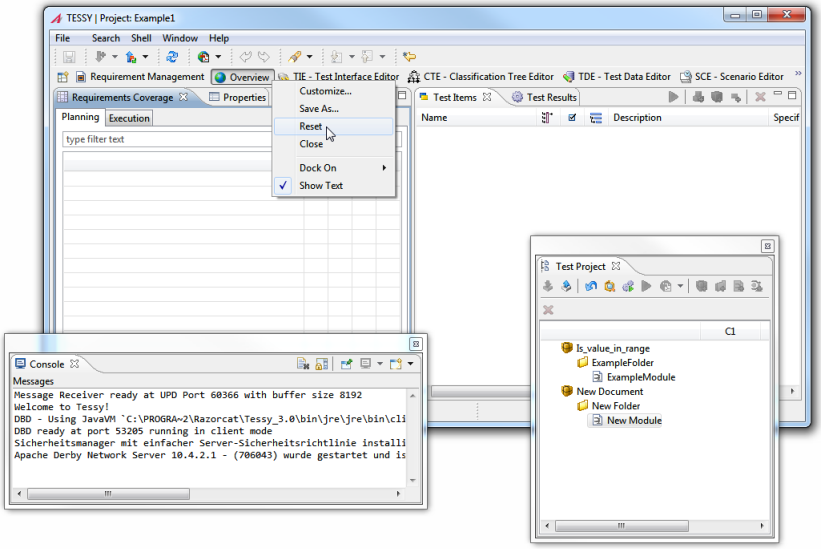


Figure 3.11: Move the views separately. To switch back, use "Reset".

Resetting workbench

To switch back all positions of perspectives and views:

→ Choose "Window" > "Reset Workbench" from the menu bar (see figure 3.12).

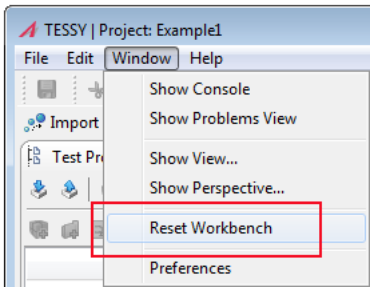


Figure 3.12: To switch back all positions of views and perspectives use "Reset Workbench".

Maximize and minimize views

You can maximize and minimize views for better visibility.

To maximize a view,

→ use the button within the right corner (see figure 3.13) or double click on the tab of the view.

Figure 3.13: Minimizing and maximizing views

The view will be maximized and other views of the perspective will be minimized, displayed by the view symbol on the left and the right (see figure 3.14).

To restore all views,

→ click on the "Restore"-button on the upper left side.

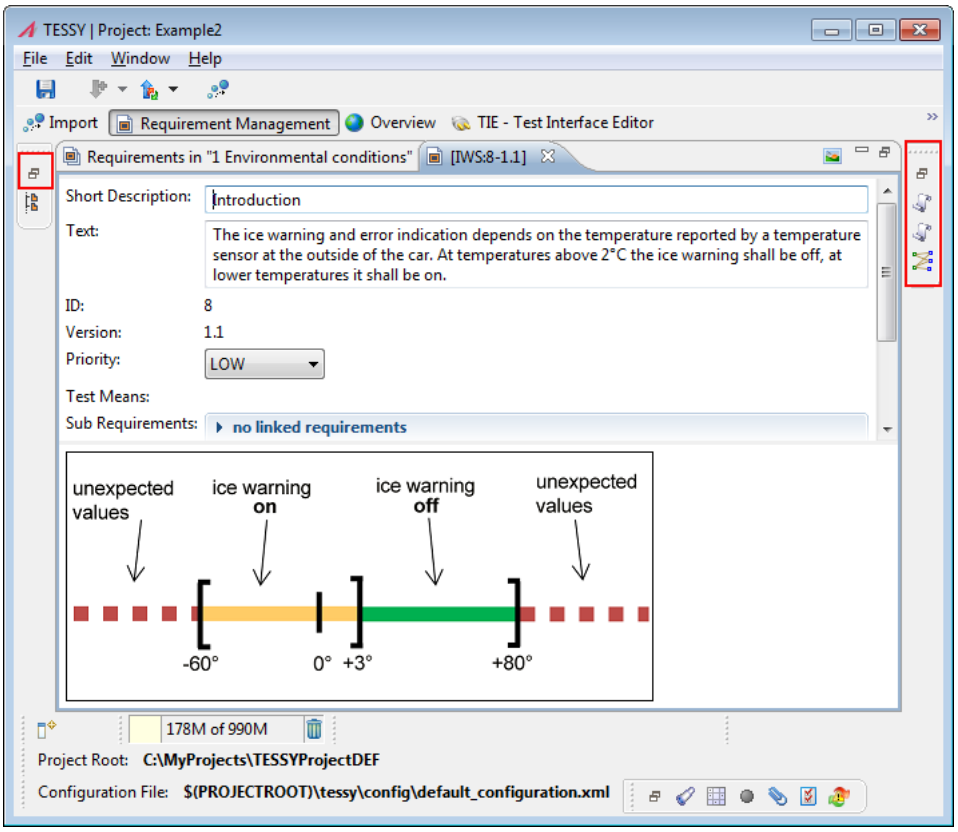
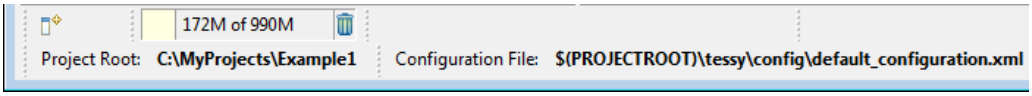


Figure 3.14: Maximized view with minimized views on the right and the restore-button on the left

There are navigation views that present hierarchical structured data. Selections on such tree items may cause other views or the editor pane to change the information being displayed.

All views are context sensitive: If you select an item within one view, another view might display other information. If something is not displayed correctly, make sure you selected the desired item.

3.2.5 Status bar



The status bar provides status information about the application and current status, e.g. the directory of the project root and the configuration file.

3.3 Using the context menu and shortcuts

3.3.1 Context menu

Most contents, tabs etc. have options that are displayed in the context menu, which is retrieved with a right click. It shows main operations as "Copy", "Paste", "Delete" etc. The context menu is context sensitive and changes as different items are selected (see figure 3.15).

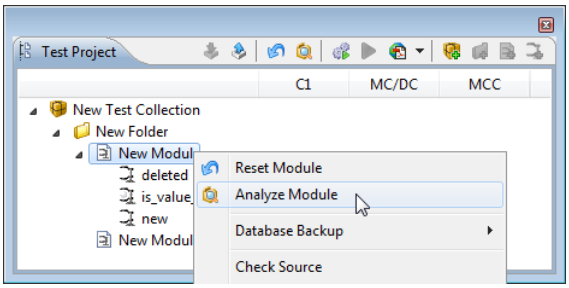


Figure 3.15: Using the context menu with right click.

3.3.2 Shortcuts

You can operate within TESSY with several shortcuts by using the keyboard:

Important: For using shortcuts make sure to be in editing mode of the current view! Otherweise the shortcut will not work.

Warning: You cannot reverse the deleting of data. Before deleting make sure this database is really not needed anymore.

Function	Shortcut / Keys	Comment
copy	Ctrl+C	
cursor positioning right	tab *alternatively* Ctrl+right arrow key	Moves the cursor to the next input section on the right side of the line. Only within TDE.

continue next page

Function	Shortcut / Keys	Comment
cursor positioning left	Shift+tab *alternatively* Ctrl+left arrow key	Moves the cursor to the input section on the left side of the line. Only within TDE.
cut	Ctrl+X	Only possible for folders, modules and (synthetic) testobjects, not for test items.
delete	Del	Only possible if the item to delete does not contain any data or folders or modules! Only manually created test cases can be deleted! You cannot delete test cases created by CTE. This prevents possible inconsistencies within the CTE document.
generate test report	Ctrl+R	
new folder	Shift+Ins	
new module	Ins	Only possible, if a test collection or folder is selected.
paste	Ctrl+V	
rename	F2	
start test execution	Ctrl+E	Executes a testobject.

Table 3.17: Shortcuts and key functions

Drag & Drop

You can also operate with "drag & drop", which is the same as "cut & paste".

Shortcuts for certain views are described in the related section within chapter 6 Working with TESSY.

4 Basic knowledge

This chapter offers a brief introduction about unit testing with TESSY and the classification tree method. It provides basic knowledge for organizing and executing a unit test in general and in particular with TESSY. The chapter about the classification tree method helps you to understand the logical system and to use the CTE.

4.1 Unit testing of embedded software 34
4.1.1 Standards that require testing 34
4.1.2 About unit testing 35
4.1.3 Considerations for unit testing 36
4.1.4 Methods for unit testing 38
4.1.5 Conclusion 40
4.2 The Classification Tree Method (CTM) 41
4.2.1 General 41
4.2.2 Steps to take 42
4.2.3 Example is_value_in_range 48

4.1 Unit testing of embedded software

Testing is an integral part of the software development process for embedded systems and its necessity to attain software quality is undisputed. This section gives a comprehensive overview on aspects of unit tests of embedded software.

4.1.1 Standards that require testing

International standards like IEC 61508 require module tests. According to part 3 of IEC 61508, the module test shall show that the module under test performs its intended function, and does not perform unintended functions. The results of the module testing shall be documented.

IEC 61508 classifies systems according to their safety criticality. There are four safety integrity levels (SIL), where 1 is the lowest level and 4 the highest, i.e. systems at level 4 are considered to be the most critical to safety. Even for applications of low criticality (i.e. at SIL 1), a module test is already "highly recommended". The tables contained in the annexes of IEC 61508, Part 3 specify the techniques that should be used, e.g. for module testing the technique "functional and black box testing" is highly recommended at SIL 1 already. Other techniques, such as dynamic analysis and testing are recommended at SIL 1 and highly recommended at SIL 2 and higher.

Part 4 of IEC 61508 defines a (software) module as a construction that consists of procedures and/or data declarations, and that can interact with other such modules. If we consider embedded software which is written in the C programming language, we can take a C-level function as a module. To prevent a mix-up between C-level functions and C source modules, we will refer to the C-level functions as units from now on.

Also other standards like the British Def Stan 00-55, ISO 15504 or DO-178B require module testing (where the nomenclature ranges from "module" to "unit" to "component"). However, all standards have more or less the same requirements for that kind of test: the tests have to be planned in advance, test data has to be specified, the tests have to be conducted, and the results have to be evaluated and documented.

4.1.2 About unit testing

What is unit testing?

During unit testing of C programs, a single C-level function is tested rigorously, and is tested in isolation from the rest of the application. Rigorous means that the test cases are specially made for the unit in question, and they also comprise of input data that may be unexpected by the unit under test. Isolated means that the test result does not depend on the behavior of the other units in the application. Isolation from the rest of the application can be achieved by directly calling the unit under test and replacing the calls to other unit by stub functions.

Unit testing tests at the interface of the unit, and unit testing does not consider the internal structure of the unit, and therefore unit testing is considered as black-box testing. The interface of the unit consists of the input variables to the unit (i.e. variables read by the unit) together with the output variables (i.e. variables written by the unit). A variable can both be an input and an output (e.g. a variable that is incremented by the unit), and the return value of the unit - if present - is always an output. The structure of a test case follows from the structure of the interface.

Unit testing is conducted by executing the unit under test with certain data for the input variables. The actual results are compared to those predicted, which determines if a test case has passed or failed.

Unit testing (of C-level functions, as described) is well suited to fulfill the requirements of module testing for IEC 61508, because unit testing is

- functional, because the functionality of the unit is tested, and
- a black-box, because the internals of the unit are not taken into account, and
- dynamic, because the test object is executed during the test.

What are the benefits?

- **Finding errors early:** Unit testing can be conducted as soon as the unit to be tested compiles. Therefore, errors inside the unit can be detected very early.
- **Saving money:** It is general knowledge that errors which are detected late are more expensive to correct than errors that are detected early. Hence, unit testing can save money.

- **Reducing complexity:** Instead of trying to create test cases that test the whole set of interacting units, the test cases are specific to the unit under test. Test cases can easily comprise of input data that is unexpected by the unit under test or by even random input test data, which is rather hard to achieve if the unit under test is called by a fully-functioning unit of the application. If a test fails, the cause of the failure can be easily identified, because it must stem from the unit under test, and not from a unit further down the calling hierarchy.
- **Giving confidence:** After the unit tests, the application is made up of single, fully tested units. A test for the whole application will be more likely to pass, and if some tests fail, the reason will have probably stemmed from the interaction of the units (and not from an error inside a unit). The search for the failure can concentrate on that, and must not doubt the internals of the units.

4.1.3 Considerations for unit testing

Which units are good test candidates?

Unit testing verifies that certain input data generates the expected output data. Therefore, units that do data processing in its widest sense, e.g. generation of data, analysis of data, sorting, making complex decisions, difficult calculations are best suited for unit testing. To find such units, the application of metrics (e.g. the cyclomatic complexity according to McCabe) may be appropriate.

Other criteria for selecting units to test may be how critical the functionality is to the unit's operation, or how often a unit is used in the application.

What is not in the scope of unit testing?

The interaction of the units is not tested during the unit test. This includes the semantic of the parameters passed between units (e.g. the physical unit of the values), and the timely relationships between units (e.g. does a unit fulfill its task fast enough to let a calling unit fulfill their tasks also at the required speed?) In addition, the interrupt behavior of the application is not in the scope of unit testing. Questions like "Does my interrupt really occur every 10 ms?" or "Which interrupt prolonged my unit unacceptably?" are not addressed by unit testing, because unit testing explicitly aims at testing the functional behavior of the unit isolated from environmental effects such as interrupts.

Why is regression testing necessary?

Regression testing is the repetition of tests that have already passed after the implementation of bug fixes or improvements in the software. Regression testing proves that a change in the software did not result in any unexpected behavior. Regression testing is a key to software quality. Obviously, the practice of regression testing requires the automation of the tests, because the effort to repeat the tests manually is too high. Even for non-repetitive unit tests, the proper tool support will save you lots of time, but tool support is indispensable for the repetition of the unit tests.

Who should conduct the tests?

The dilemma: It is commonly accepted that a software developer is badly suited to test his own software, especially if the complete implementation, or the compliance of the implementation with the specification is an issue (blindness against own faults). If the developer has forgotten to implement a certain functionality, it is likely he will also forget a test that will reveal the missing functionality. If the developer has misinterpreted the specification, it is likely that his tests will pass in spite of the wrong functionality.

On the other hand, experience has shown that a tester, who should test a code not written by him must put a lot of effort into understanding the function´s interface. The tester must find out the meaning of the variables, and which values to use to conduct certain tests. E.g., if the test specification requires the test of something "green", which variable (or variables) represents the color, and which value of the variable represents green? The prediction of the expected results poses similar problems.

If the developer does not do tests, this gives rise to additional efforts, because the failed test has to be passed to the developer, he has to reproduce the failure, correct the problem, and then normally a concluding external regression test has to take place. Furthermore, additional effort rises due to the fact that the developer will not hand out his software to the QA department without having done at least some tests. This duplicated test effort could be saved if the developer immediately starts testing by using the externally predefined test cases.

The way out: A way out of that dilemma could be that a tester, who has not written the code, specifies the test cases according to the functional specification of the unit, including the expected results. He can use abstract data for this (e.g. color = green). The set of test cases is handed over to the developer of the software. For him, it should be no problem to set the input variables to the required values (e.g. the appropriate RGB value for green). If a test fails, the developer can immediately correct the problem and re-run all tests that have passed so far (regression testing). Testing is seen as an additional step during the implementation of software, in comparison to the compiling step, where the

compiler finds all syntax errors, and the developer corrects them interactively, verifying his changes by subsequent compiler runs.

However, standards require the organizational separation of development and test, due to the initial mentioned reason of blindness against own faults. Possibly, it could be sufficient to only separate the specification of the test cases from the development, and to consider the conduction of predefined test cases not to suffer under the above mentioned blindness.

Furthermore, a developer´s time is often considered as too valuable to be wasted on testing, which is why developer testing is not found often in practice. However, this is going to be reconsidered.

What is special for testing embedded software?

For embedded software it is essential that the unchanged source code with all the non-ANSI keywords and non-ANSI peculiarities is used for testing. For instance, some cross compiler for embedded systems allow for bit fields that are smaller than the integer size, e.g. 8-bit wide bit fields in a 16-bit application. This is forbidden by the ANSI C standard, but justifiable by the perfect adaptation to the embedded system. Naturally, the unit test results are worthless, if this illegal size cannot be maintained during the tests. This requires specialized tools. Furthermore, it is also essential that the concluding tests at least execute on the actual hardware, i.e. the embedded microcontroller. This is a challenge, but there are ways to attenuate this. Using a cross compiler for the microcontroller in question is a prerequisite, preferably the exact version that will be used also for the user application.

4.1.4 Methods for unit testing

Unit test tools can follow two technical approaches towards unit test: The test application approach uses a special application for conducting the unit tests. This is the usual approach. The original binary test uses the unchanged user application for testing.

a. Test application

The usual method for unit test tools to conduct unit tests is to generate a test driver (also called test harness) and compile the test driver together with the source code of the unit under test. Together they form the test application. The test driver includes startup code for the embedded microcontroller, the main() function entry, and a call to the unit under test. If required, the test driver contains also code for stub functions and the like. For each unit to test, an own test application is created. This test application is

used to conduct the unit tests. For that, the test application is loaded into an execution environment capable of executing the test application. This execution environment is normally a debugger connected to an (instruction set) simulator, an in-circuit emulator stand-alone or connected to a target system, a JTAG or BDM debugger or the like. After test data is transferred to the execution environment, (the test data may already be included in the test application), tests are conducted and the results are evaluated.

To execute the test application on the actual hardware, the test application must not only be compiled using a cross compiler for the microcontroller in question, but also the test application must fit into the memory present on the actual hardware. Also, the startup code of the test application must take into account peculiarities of the actual hardware, e.g. the enabling of chip selects and the like. Making the test application fit into memory can be simplified by using an in-circuit emulator, which provides emulation memory, and serves as a kind of generalized hardware platform for the microcontroller in question.

When the actual hardware has to be used and if memory on this hardware is very limited, the test application must be minimized to fit into this memory. This is especially challenging for single chip applications, where only the internal memory of the microcontroller is available. If test data is included in the test application (and memory is limited), a single test application can only include a few test cases, which in turn means several test applications for the test of one unit, which is cumbersome. An approach which avoids this, keeps the test data separated from the test application, which allows not only for a minimized test application, but also allows you to change the test data without having to regenerate the test application.

b. Original binary test

Another approach is to use the unchanged user application for unit testing. This resembles the manual test that is usually done by a developer after the application is completed. The complete application is loaded into the execution environment, and the application is executed until the unit to be tested is eventually reached. Then the input variables are set to the required values, and the test is conducted.

Pros and cons

The advantage of the Original Binary Test approach is that the unit under test is tested exactly in its final memory location. There is no extra effort (or hassle) for compiling and linking a test application, because the user application is used, which is already compiled and linked or had to be compiled and linked anyway. Because the user application must fit in the memory anyway, problems regarding the size of the application can be neglected. Even applications that already reside in the ROM of the hardware can be tested. Even

if the cross compiler used to compile the user application is no longer at hand, tests are still feasible.

However, this Original Binary Test approach has some disadvantages compared to using a test application:

- There is no control over the test execution. It depends on the user application, when the unit under test is reached. It may be the case that the unit under test is never reached, or only after some special external event has happened, e.g. the push of a button of the actual hardware and an interrupt resulting from this.
- During the Original Binary Test, stub functions cannot be used. This is clear because the application is already linked using the current functions that are called by the unit under test. A unit is always tested using the other units of the application. Therefore, the unit under test is not isolated from the rest of the application, and errors of called units may show up during the test of the unit under test.
- It is not possible to use arbitrary test data for the unit test. For instance, if the unit under test gets its test data by a pointer pointing to a memory area, the amount test data must fit into this memory area, which was allocated by the user application.

Apart from its easy usage, which possibly could be the only means to do some unit testing at all, the Original Binary Test has strong disadvantages, which are essential for proper unit testing and therefore one could even insist that it is not a unit test in its strictest sense.

4.1.5 Conclusion

Besides being required by standards, unit testing reduces the complexity of testing, finds errors early, saves money, and gives confidence for the test of the whole application. If used in the right way, unit testing can reduce development/test time and therefore reduce the time-to-market. To conduct regression tests, test automation is indispensable. This requires tool support.

4.2 The Classification Tree Method (CTM)

The objective of the Classification Tree Method (CTM) is to transform a (functional) definition of a problem systematically into a set of error-sensitive, low redundancy set of test case specifications. This document gives a comprehensive overview of the CTM.

4.2.1 General

Testing is a compulsory step in the software development process. The planning of such testing often raises the same questions:

- How many tests should be run?
- What test data should be used?
- How can error-sensitive tests be created?
- How can redundant tests be avoided?
- Have any test cases been overlooked?
- When is it safe to end testing?

Anyone who has been confronted with such issues will be glad to know that the CTM offers a systematic procedure to create test case specifications based on a problem definition.

The objective of the CTM is to transform a (functional) definition of a problem systematically into a set of error-sensitive, low redundancy set of test case specifications. The systematic approach yields a high probability that the resulting set of test specifications is complete and no relevant tests are overlooked. Naturally, correct usage of the method and an appropriate integration in the development process are prerequisites. Having a complete set of tests gives evidence when it is safe to end testing.

The CTM is applied by a human being. Therefore, the outcome of the method depends on the experiences, reflections, and appraisals of the user of the CTM. Most probably two different users will come out with a different set of test case specifications for the same functional problem. Both sets could be considered to be correct, because there is no absolute correctness. It should be clear that there are set of test cases that are definitively wrong or incomplete. Because of the human user, errors cannot be avoided. One remedy is the systematic inherent in the method. This systematic guides the user and stimulates his creativity. The user shall specify test cases with a high probability

to detect a fault in the test object. Such test cases are called error-sensitive test cases. On the other hand, the user shall avoid that too many test cases are specified, that are superfluous, i.e. do not increase test intensiveness or test relevance. Such test cases are called "redundant" test cases. It is advantageous, if the user is familiar with the field of application the method is applied in.

The CTM is a general method: It can not only be applied to module/unit testing of embedded software, but to software testing in general and also to functional testing of problems, that are not software related. The prerequisite to apply the method is to have available a functional specification of the behaviour of the test object. The CTM incorporates several well-known approaches for test case specification, e.g. equivalent partitioning, and boundary value analysis.

The CTM stems from the former software research laboratory of Daimler in Berlin, Germany.

4.2.2 Steps to take

a. Defining the functional problem

The first step is to describe the expected behaviour of the test object, e.g. "If the button is pushed, the light will go on; if the button is released, the light will go off". Data processing software normally solves functional problems, since input data is processed according to an algorithm (the function) to become output data (the solution).

b. Determining the test-relevant aspects

Analyse the functional specification. This means, you think about this specification with the objective to figure out the test-relevant aspects of the specification. An aspect is considered relevant if the user expects that aspect to influence the behaviour of the test object during the test. In other words, an aspect is considered relevant if the user wants to use different values for this aspect during testing. To draw the tree, these aspects are worked on separately. This reduces the complexity of the original problem considerably, what is one of the advantages of the CTM.

Example for a test-relevant aspect

Consider systems that measures distances in a range of some meters, e.g. the distance to a wall in a room. Those systems usually send out signals and measure the time until they receive the reflected signal. Those systems can base on two different physical effects: One can use sonar to determine the distance, whereas the other can use radar.

The question is now: Is the temperature of the air in the room a test relevant aspect for the test of these measurement systems? The answer is yes for one system and no for the other:

The speed of sound in air (sonar) is dependent on the temperature of the air. Therefore, to get exact results, the sonar system takes this temperature into account during the calculation of the distance. To test if this is working correct, you have to do some tests at different temperatures. Therefore, the temperature is a test-relevant aspect for the sonar system. On the other hand we all know that the speed of a radar signal, that travels at the speed of light, is independent from the temperature of the air it travels in (it did not even need air to travel). Therefore, the temperature of the air is not a test-relevant aspect for the testing of the radar system. It would be superfluous to do testing at different temperatures.

This example shows that it needs careful thinking to figure out (all) test relevant aspects. It would lead to poor testing if someone simply takes the test cases for the radar system and applies them to the sonar system without adding some temperature-related test cases. Additionally, this example illustrates that it is advantageous to have some familiarity with the problem field at hand when designing test cases.

c. Classifying the values of a test-relevant aspect

After all test relevant aspects are determined, the values that each aspect may take are considered. The values are divided into classes according to the equivalence partitioning method: Values are assigned to the same class, if the values are considered equivalent for the test. Equivalent for the test means that if one value out of a certain class causes a test case to fail and hence reveals an error, every other value out of this class will also cause the same test to fail and will reveal the same error.
In other words: It is not relevant for testing which value out of a class is used for testing, because they all are considered to be equivalent. Therefore, you may take an arbitrary value out of a class for testing, even the same value for all tests, without decreasing the value of the tests. However, the prerequisite for this is that the equivalence partitioning was done correctly, what is in the responsibility of the (human) user of the CTM.

Please note:

- Equivalent for the test does not necessarily mean that the result of the test (e.g. a calculated value) is the same for all values in a class.
- Equivalence partitioning must be complete in mathematical sense: Every possible value of a test relevant aspect must be assigned to a class.
- Equivalence partitioning must be unique in mathematical sense: A value of a test relevant aspect must be assigned to a single class, and not to several classes.

Example for equivalence partitioning: Ice Warning

An ice warning indication in the dashboard of a car shall be tested. This ice warning indication depends on the temperature reported by a temperature sensor at the outside of the car, which can report temperatures from -60°C to +80°C. At temperatures above 3°C the ice warning shall be off, at lower temperatures it shall be on.

It is obvious that the temperature is the only test-relevant aspect. To have an reasonable testing effort, we do not want to have a test case for every possible temperature value. Therefore, all possible temperature values need to be classified according to the equivalence partitioning method.
It is best practice to find out if invalid values may be possible. In our case a short circuit or an interruption of the cable could result in an invalid value. Therefore, we should divide the temperature in valid and invalid values first. The invalid values can relate to temperatures that are too high (higher than 80°C) and to ones that are too low (lower than -60°C). It is tempting to form two classes out of the valid temperatures: The first class shall contain all values that result in the ice warning display being on (from -60°C to 3°C) and the other class shall contain all values that result in the ice warning display being off (from 3°C to 80°C).

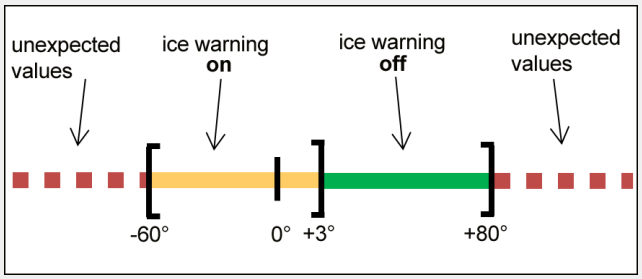


Figure 4.1: Initial equivalence partitioning for "ice warning"

The equivalence partitioning in the figure above leads to at least four test cases, because we need to take a value out of each class for the tests.

d. Repeating equivalence partitioning

An equivalence class can be sub-divided according to additional aspects. This equivalence partitioning on several levels reduces the complexity of equivalence partitioning, because you can consider each class isolated from the other classes and decide, if and how it needs to be sub-divided or not. Furthermore, this equivalence partitioning on several levels documents the thoughts resp. stages of work until the final equivalence partition. This serves understandability and traceability of the result. Also it allows easily reverting steps if the final equivalence partition has become too fine granulated.

Example for repeated equivalence partitioning

For the example ice warning, the classification of the valid values is not detailed enough, because according to the equivalence partitioning method, it would be sufficient to use a single, arbitrary value out of a class for all the tests. This could be for instance the value 2°C out of the class of temperatures, for which the ice warning display is on. In consequence, no test with a minus temperature would check if the ice warning display is on. To avoid this consequence, you could divide this class further according to the sign of the temperature:

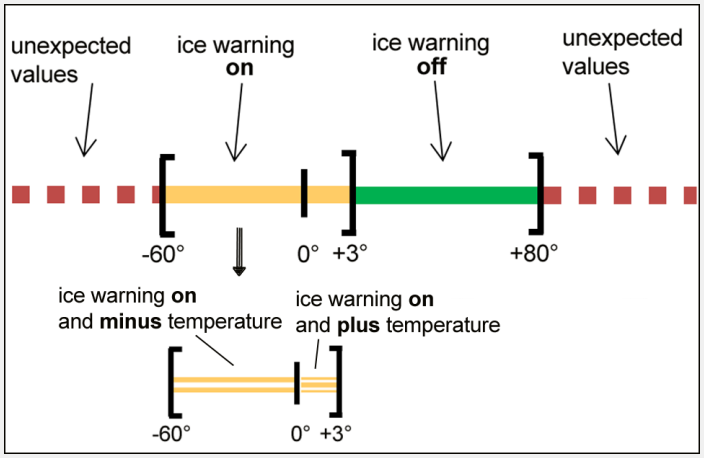


Figure 4.2: Repeated equivalence partitioning for "ice warning"

Result: Classification tree

Using the CTM, the result of the repetition of equivalence partitioning for all test relevant aspects is depicted in the CT. The root represents the functional problem, the test relevant aspects. Test relevant aspects (classifications) are drawn in nodes depicted by rectangles. Classes are ellipses.

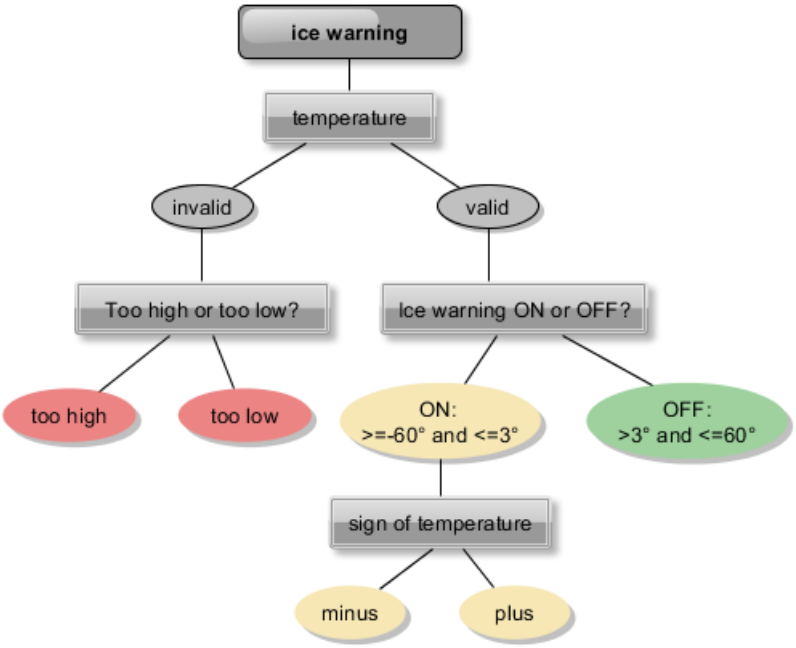


Figure 4.3: A possible CT for "ice warning"

e. Using boundary values

The idea behind using boundary values is that values at the borders of a range of values are better suited to form error-sensitive test cases than values in the middle. The idea behind boundary values analysis is contrary to equivalence partitioning, because one method takes a set of values as equivalent and the other method prefers special values in such a set. Despite the fact that the idea behind boundary values analysis is exactly the opposite of equivalence partitioning, both approaches can be expressed in the CTM.

f. Testing a hysteresis

The current problem specification of the ice warning-example does not mention hysteresis. It may be tempting to extend the current problem specification in that fast changes in the state of the ice warning display shall be avoided. For instance, the ice warning display shall be switched off only after the temperature has risen to more than 4°C. This could be realized by a hysteresis function. The necessary test cases for such a hysteresis function can be specified by the CTM.

g. Specifying test cases

Test cases are specified in the so-called combination table below the CT. The leaf classes of the CT form the head of the combination table. A line in the combination table depicts a test case. The test case is specified by selecting leaf classes, from which values for the

test case shall be used. This is done by the user of the method, by setting markers in the line of the respective test cases in the combination table.

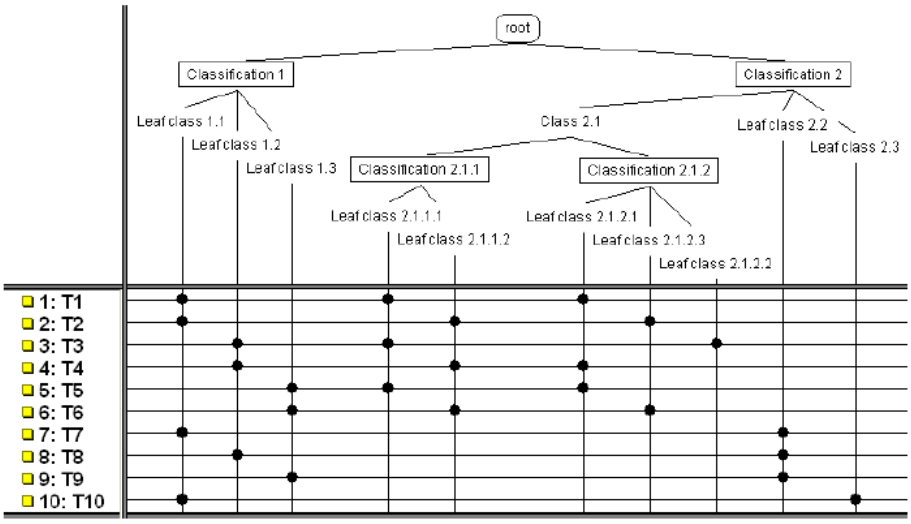


Figure 4.4: Result of the CTM: tree (above) with combination table (below)

It may be tempting to combine every class with every other class during the specification of the test cases. Besides the fact, that not every combination might be possible for logical reasons, it is not the intention of the CTM to do so, it could be done automatically by a tool. This would lead to many test cases, with the disadvantages of loss of overview and too much effort for executing the test cases.

The objective of the CTM is to find a minimal, non-redundant but sufficient set of test cases by trying to cover several aspects in a single test case, whenever possible. Similar to the drawing of the tree, it depends on the appraisal and experience of the user of the method, how many and which test cases are specified.

Obviously the size of the tree influences the number of test cases needed:
A tree with more leaf classes naturally results in more test cases than a tree with less leaf classes. The number of leaf classes needed at least for a given tree is called the *minimum criterion*. It can be calculated from the consideration that each leaf class should be marked in at least one test case, and that some leaf classes cannot be combined in a single test case, because the classes exclude each other.
Similar a *maximum criterion* can be calculated, which gives the maximal number of test cases for a given CT. A rule of thumb states that the number of leaf classes of the tree gives the order of magnitude for the number of test cases required for a reasonable coverage of the given tree.

4.2.3 Example is_value_in_range

Problem definition:

A start value and a length define a range of values. Determine if a given value is within the defined range or not. Only integer numbers are to be considered.

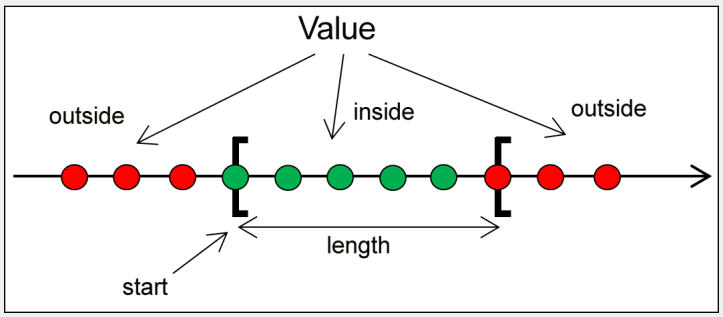


Figure 4.5: The problem "is_value_in_range" depicted graphically

It is obvious, that completed testing is practically impossible, because we get 65536 * 65536 * 65536 = 281.474.976.710.656 test cases, even if we assume only 16 bit integers. If we would assume 32 bit integers . . . well, we better do not.

Test-relevant aspects

The start of the range and the length can be regarded as test relevant aspects. This is convenient since, according to the problem definition, a range of values is defined by a start value and a length. It reflects the intention to use different values for the start and the length during testing.

We should have some test cases, which result in inside, and other test cases which result in outside. We call the corresponding aspect *position*, because the position of the value under test with respect to the range determines the result. So the three test-relevant aspects to be used for classifications are initial *value*, *length* and *position* and they thus form the basis of the CT:

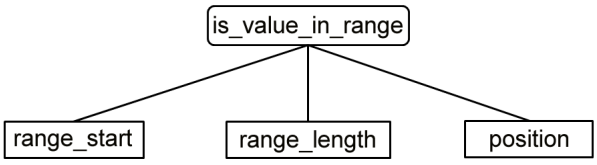


Figure 4.6: The initial CT with three test-relevant aspects

Forming classes

Now classes are formed for the base classifications according to the equivalence partitioning method. Usually, the problem specification gives us hints how to form the classes. E.g. if the problem specification would state: "If the start value is greater than 20, the length value doubles", then we should form a class for start values greater than 20 and a class for start values smaller or equal to 20.

Unfortunately, the problem specification at hand is too simple to give us similar hints. However, since the start value can take on all integer numbers, it would be reasonable to form a class for positive values, a class for negative values, and another class for the value zero. It would also be reasonable to form just two classes, e.g. one class for positive start values including zero and the other class for negative start values. This depends on ones emphasis having zero as value for the start of the range in a test case or not.

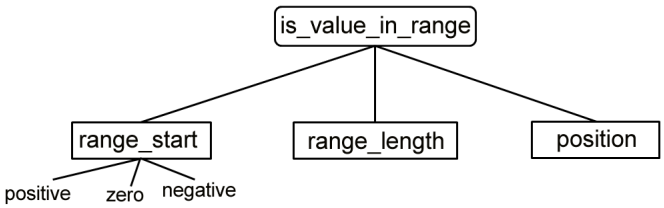


Figure 4.7: The CT for is_value_in_range, 2nd step

Because of the systematic inherent in the CTM, and because *range_length* is an integer as well as *range_start*, it is stringent to use for *range_length* the same classes as for *range_length*. This results in the following tree:

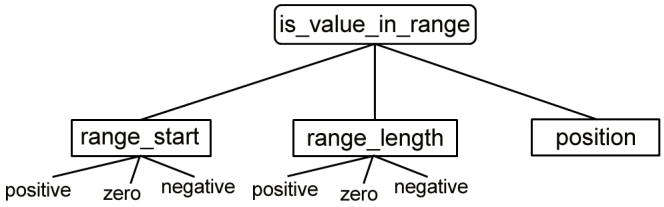


Figure 4.8: The CT for is_value_in_range, 3rd step

A first range specification

To specify a first range (to be used in the first test case), we have to insert a line in the combination table and to set markers on that line:

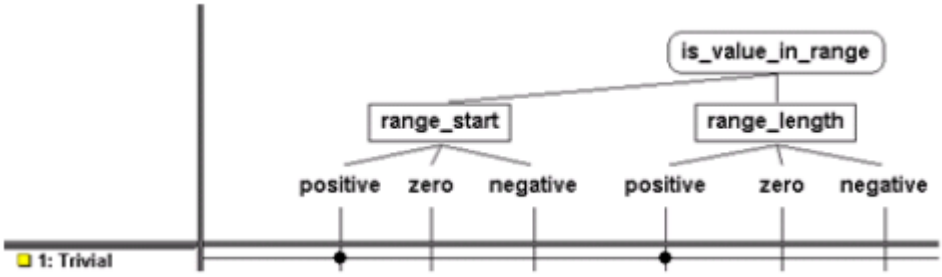


Figure 4.9: A first specification for the range in the combination table

Two markers are set on the line for the first specification. One marker selects the class *positive* for the start of the range. The other marker selects the class *positive* for the length of the range. A range with the start value of, say, 5 and a length of 2 would accord to the specification. This first specification was named *trivial*.

A second range specification

We can insert a second line in the combination table and specify a much more interesting tests case:

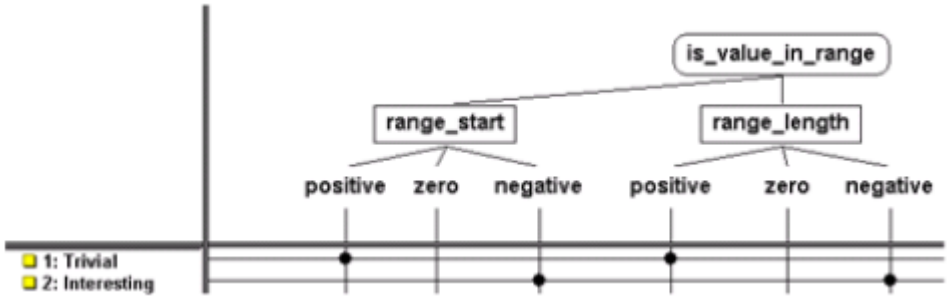


Figure 4.10: A second specification for the range in the combination table

For the second specification again two markers are set. They specify that a negative value shall be used both for the start and for the end of the range. Hence a range with the start value of -5 and a length of -2 would accord to the second specification. But this value pair raises some questions: Shall the value -6 lie inside the range? Or shall the value -4 lie inside the range? Or shall no value at all lie inside the range, if the length of the range is negative? Each opinion has its supporters and it is hard to decide what is to be considered correct. Actually, at this point it is out of our competence to decide what is correct. We have found a problem of the specification!

It is a valuable result to find a problem (omission or contradiction) in the functional problem specification, and that it was achieved in the case during test case specification for the functional problem. It is generally more likely to detect a problem in the functional specification if the test case specification is systematic. The CTM is a systematic method for test case specification. Hence, the CTM provides good means to detect problems in the functional problem specification.

Probably a test case using a negative length would not have been used if the test case specification would have been done spontaneous and non-systematic. But a negative length is completely legal for the functional problem specification that was given above. If you consider that the problem specification at hand was a very simple one, you may imagine how likely it is to overlook a problem in a more comprehensive and complicated problem specification.

Extending the tree by a boundary class

In case we are not satisfied with the fact that a fixed single positive value, e.g. 5, may serve as value for the start of the range in all test cases, we can sub-divide the class *positive* according to a suitable classification. In our example, we classify according to the size. The idea behind this is to have a class containing only a single value, in our case the highest positive value existing in the given integer range. We use this value because it is an extreme value, and as we know, using extreme values (or boundary values) in test cases is well-suited to produce error-sensitive (or interesting) test cases.

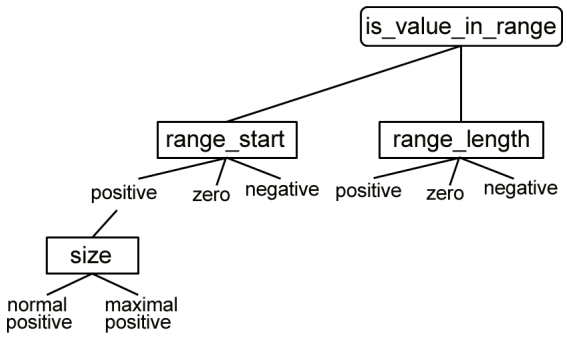


Figure 4.11: The CT for is_value_in_range, 4th step

In the figure above, the positive values for the start of the range are subdivided according to their size. This results in the two classes *normal positive* and *maximal positive*. The class *maximal positive* holds the highest possible positive value (i.e. MAX_INT), and

the class *normal positive* holds all other positive values. This satisfies mathematical completeness.

Remark 1: Another possibility to classify the positive start values would have been for instance to classify in odd and even values. This would have been completely legal. This would have been probably also sensible for e.g. a problem of number theory, but not target-oriented for the problem at hand.

Remark 2: Please note that for the moment we do not know and we need not to know the size (in bits) of the integers used in the problem at hand. We simply specify "the highest positive value in the given integer range". This keeps our test case specification abstract! I.e. our test case specification is appropriate for any integer size. As soon as we assume we use e.g. 16 bit integers, and therefore parameterize our test case by specifying 32767 as value in the class *maximal positive*, we loose this abstraction. I.e. if we port the parameterized test case to a 32 bit integer system, the test case looses its sense. This is not the case if we port the abstract test case specification.

Another interesting test case specification

With the CT extended according to figure 4.11 The CT for is_value_in_range, 4th step, we can insert an additional line in the combination table and specify again an interesting range for a third test case:

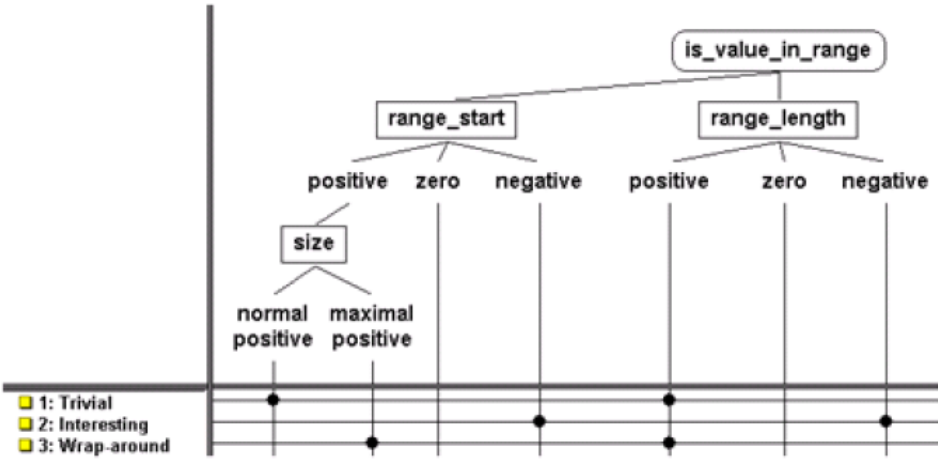


Figure 4.12: The third range specification provokes a wrap-around

The third range specification in the figure above combines the highest positive number for the start value of the range with a positive length, i.e. the range exceeds the given integer range.

The situation with the third range specification is similar to the situation depicted in the figure above. The situation raises some questions: Will the situation be handled sensible and gracefully by the test object? Or will it crash due to the overflow? Will the negative values on the left hand side ibe accounted to lie inside the range or not? And what is correct with respect to the last question? The problem specification above does not give an answer to the latter question, again we have found a weak point in the problem specification.

To sum up, designing test cases according to the CT method has revealed two problems of the problem specification and has lead to interesting test cases so far.

The completed classification tree

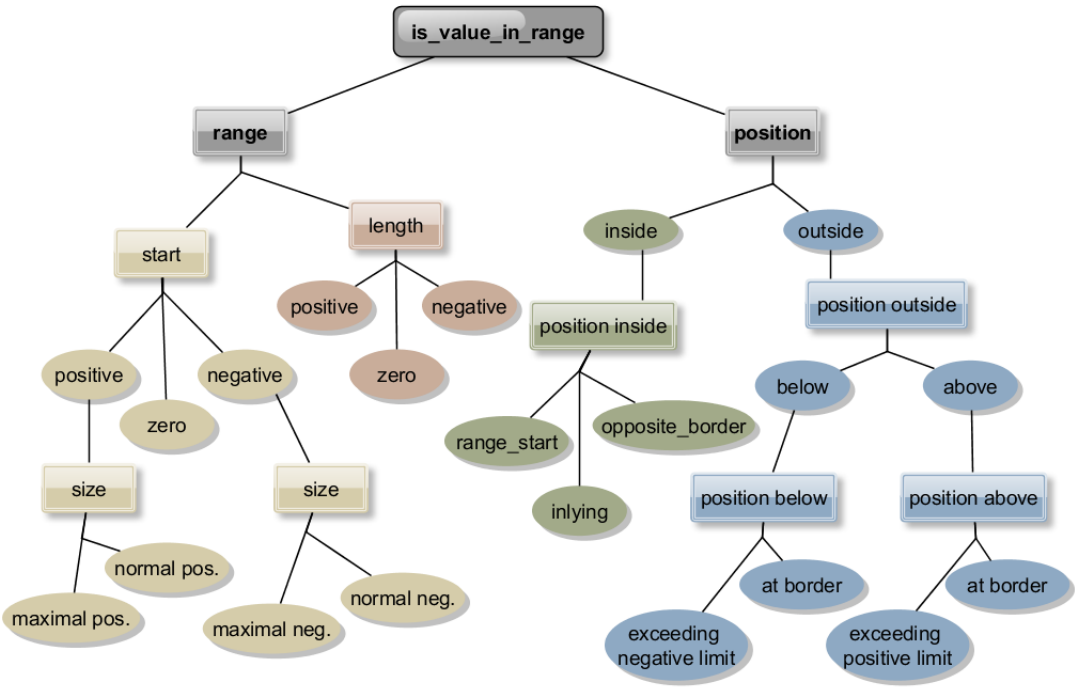


Figure 4.13: The completed CT for is_value_in_range

In the figure above, one possible completed CT is depicted. Classifications are depicted by rectangles, classes by ellipses. The "range" node is a composition with two classifications as child elements. This tree is discussed in the following:

- Analogous to the class *maximal positive* for the start value of the range, a class *maximal negative* is introduced. The idea behind this class is to combine the maximal negative start value with a negative length of the range, what shall provoke an underflow or negative wraparound. This idea comes from the systematic in the CTM: If a positive wrap-around is seen as an interesting test case, also a negative wrap-around should be exercised.

- An example for a composition is given by *range*. A composition may be used for a relation "consists of". In our case, the range consists of a *start* value and a *length*.

- The final tree features still the three initial classes *positive*, *zero*, and *negative* for the length of the range. It is important to note that the tree reveals at a glance that nothing like *maximal positive* length or similar is considered to be useful for the testing problem at hand.

- It is obvious that a position can either be *inside* or *outside* the range, hence this classification suggests itself. Furthermore, it is obvious that there are two different areas outside the range: *below* the range and *above* the range. This is reflected in the classification *position outside*. (If the tree would miss such a classification, it may well be considered incorrect).

- The class *inside* of the classification *position* could well be a leaf class of the classification tree. However, in the CT in the figure above, this class is subdivided further in the sub-classes *range_start*, *opposite_border*, and *inlying*. This is done to force the use of boundary values in the test cases. If a test case specification selects the class *range_start*, the value that shall be checked if it is inside the range or not shall take the value of the start of the range, that is the lowest value that is considered to be inside the range, a boundary value. The class *opposite_border* is intended to create an analogous test case specification, but using the highest value that is considered to be inside the range. The class *range_start* and the class *opposite_border* both contain only a single value. All other values inside the range are collected in the class *inlying*; this class exists mainly because of the requirement for completeness of equivalence partitioning. A similar approach to use boundary values is visible in the classes *at border* for positions outside the range.

The completed test case specification

In the next figure, the same CT is depicted with a completed combination table, what results in a complete test case specification for the functional problem:

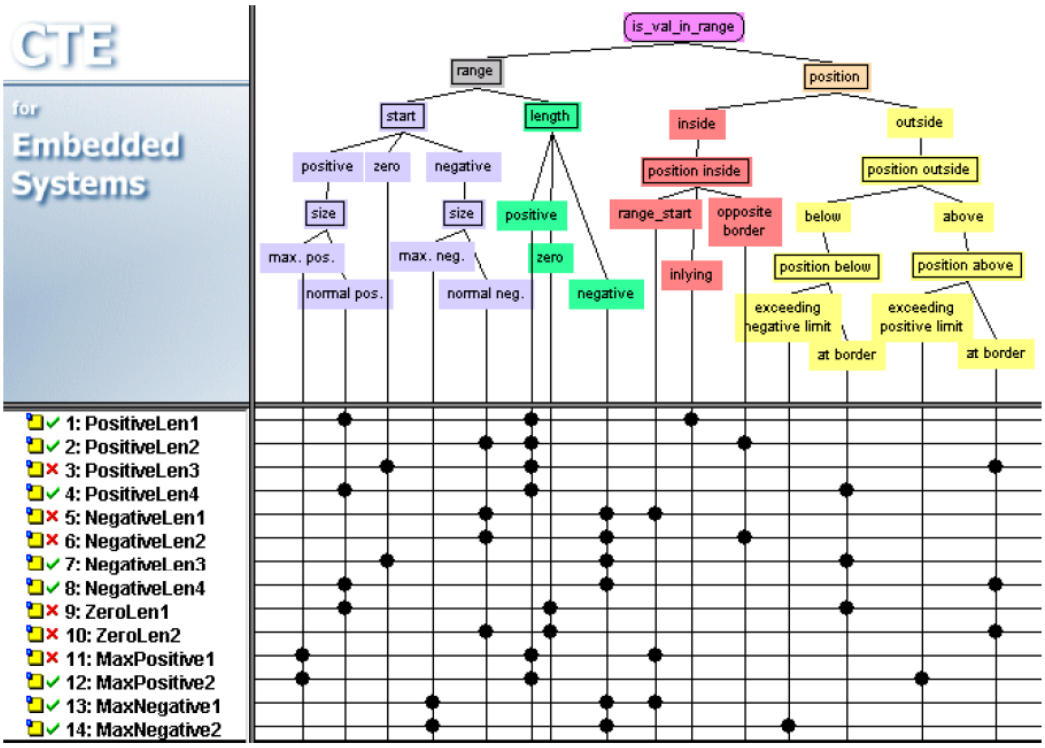


Figure 4.14: The completed test case specification

The test case specification above lists 14 test cases. Please note that these are specified by the user and depend on its judgment. Based on the CT it is possible for some values to be determined that provide clues to the number of test cases required.

The first value is the number of test cases, if each leaf class is included at least once in a test case specification. This number is known as the minimum criterion. In our example, the largest amount of leaf classes, namely seven, belong to the base classification *position*. Seven is thus the value of the minimum criterion. The maximum criterion is the number of test cases that results when all permitted combinations of leaf classes are considered.

In our example, the maximum criterion amounts to 105 (i.e. 5 * 3 * 7). The maximum criterion takes into account that it is not possible to select e.g. a negative length and a positive length for the same test case specification, because this is impossible by the construction of the tree. The maximum criterion takes not into account that it is not

possible to select e.g. a zero length and *inlying*, because this is not impossible by the construction of the tree, but by the semantics of the function problem.

A reasonable number of test case specifications obviously lies somewhere between the minimum and the maximum criterion. As a rule of thumb, the total number of leaf classes gives an estimate for the number of test cases required to get sufficient test coverage. In the test case specification, the CT has 15 leaf classes, what fits well to 14 test cases.

By the test case specification in the figure above, you can deduct how the functional problem specification was extended with respect to the questions raised in sections "A second range specification" and "Another interesting test case specification":

- "If the length of the range is negative, are there values that can be inside the range?" The answer is "yes", because in test case specification no. 5 and no. 6 a negative length shall be used and the position of the value shall be inside the range.
- "If the length of the range exceeds the given integer range, shall negative values be inside the range?" Test case specification no. 12 clarifies that this should not be the case.

The leaf class *inlying* is selected for only one test case specification (no. 1). This reflects the fact that this class exists only because of the requirement for mathematical completeness of equivalence partitioning, and not because the inlying values are considered to produce error-sensitive test cases.

Another test case specification

Here is an alternative test case specification to the functional problem specification at hand depicted:

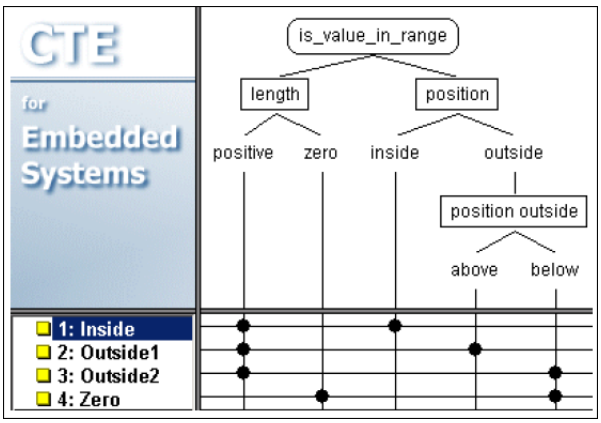


Figure 4.15: An alternative test case specification

What are the differences to the more elaborated test case specification in the section above?

- The start value of the range is not mentioned in the CT. This means, the start value is not considered to be a test-relevant aspect by the user of the CTM. In consequence, any arbitrary value can be used as start value in the four test cases. This value can be fix for all test cases, but does not have to be.
- The problem of a negative length is completely neglected. For the problem specification from section Problem which specifies a length to be an integer and hence also the length to be negative, this is a serious flaw.
- The problem of wrap-around is neglected. This may be considered to be an esoteric problem, and therefore it could be accepted that it is not mentioned in the alternative test case specification.
- The usage of boundary values is not forced by the alternative test case specification. This is questionable, because boundary values produce error-sensitive test cases. The alternative test case specification minimizes testing effort (by specifying only four test cases), but this is at the cost of thoroughly testing.

But the point is not which test case specification is better. The main point is:

Test case specification according to the CTM visualizes testing ideas!

5 Practical exercises

This chapter will show you how to work with TESSY on the basis of prepared exercises:

- The Quickstart 1: Exercise is_value_in_range is a very basic example of how to handle with TESSY.
- The Quickstart 2: Exercise Classification Tree Editor (CTE) gives a short and easy introduction of handling with the Classification Tree Editor (CTE). It continues the Quickstart 1.

5.1 Quickstart 1: Exercise is_value_in_range . . . 61
5.1.1 Creating a test project . . . 62
5.1.2 Specifying the target environment . . . 64
5.1.3 Adding the test object and analyzing the C-source file . . . 65
5.1.4 Editing the test object interface . . . 68
5.1.5 Designing test cases . . . 69
5.1.6 Adding test cases and test steps . . . 69
5.1.7 Entering test data . . . 71
5.1.8 Executing the test . . . 74
5.1.9 Repeating the test run with coverage instrumentation . . . 75
5.1.10 Analyzing the coverage . . . 76
5.1.11 Creating a report . . . 81
5.1.12 Repeating the test run with requirements . . . 83
5.1.13 Reusing a test object with a changed interface . . . 93
5.2 Quickstart 2: Exercise Classification Tree Editor (CTE) . . . 101
5.2.1 Creating classifications . . . 102
5.2.2 Creating classes . . . 104
5.2.3 Entering test data . . . 104
5.2.4 Creating test cases . . . 107

5.1 Quickstart 1: Exercise is_value_in_range

In this exercise we will get to know the basic functionality of testing with TESSY. We will operate with the example "is_value_in_range" which will give you a fast introduction and an overview as well as the terms of importance.

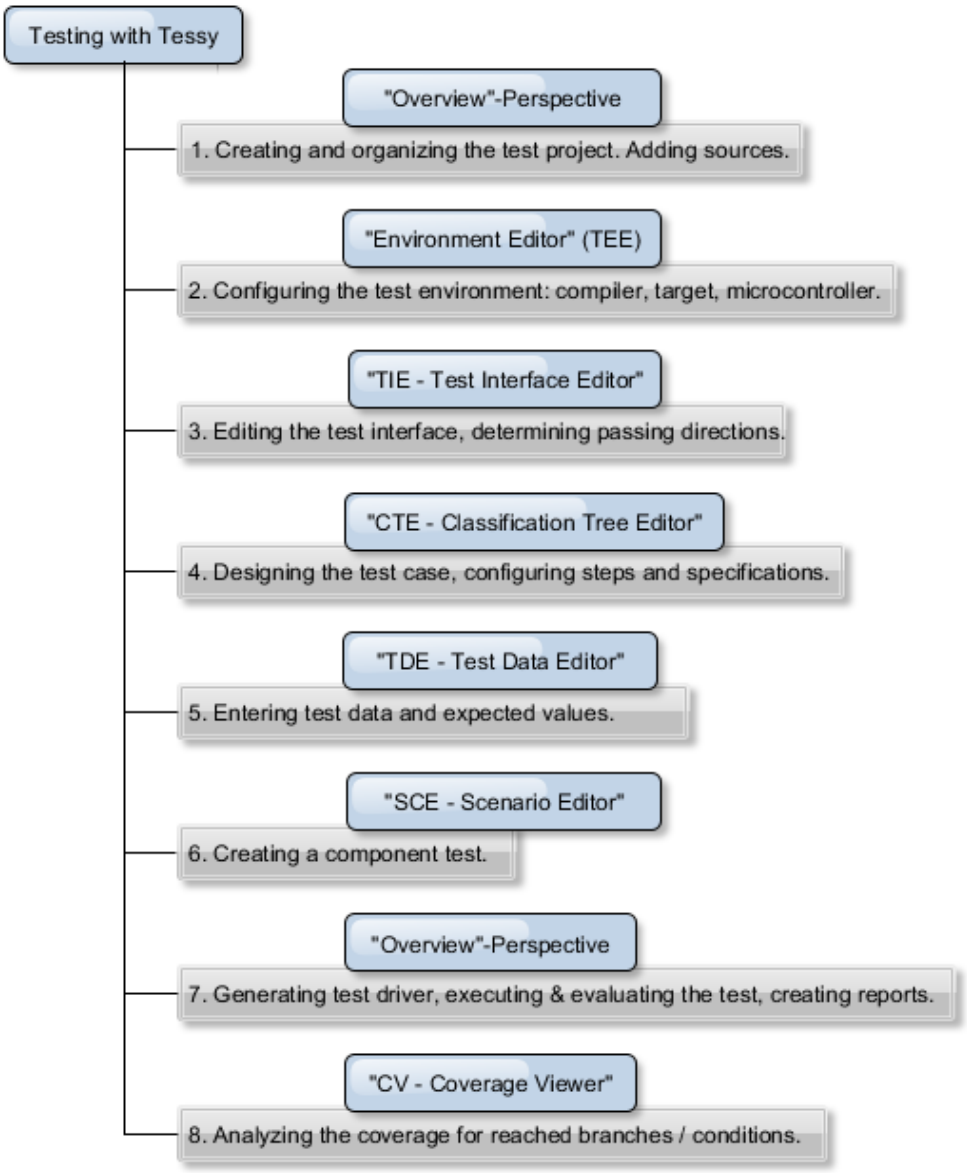

Testing with Tessy
"Overview"-Perspective
1. Creating and organizing the test project. Adding sources.
"Environment Editor" (TEE)
2. Configuring the test environment: compiler, target, microcontroller.
"TIE - Test Interface Editor"
3. Editing the test interface, determining passing directions.
"CTE - Classification Tree Editor"
4. Designing the test case, configuring steps and specifications.
"TDE - Test Data Editor"
5. Entering test data and expected values.
"SCE - Scenario Editor"
6. Creating a component test.
"Overview"-Perspective
7. Generating test driver, executing & evaluating the test, creating reports.
"CV - Coverage Viewer"
8. Analyzing the coverage for reached branches / conditions.

Figure 5.1: Operational sequences in TESSY

Central test activities

A unit test in TESSY is divided into the following central test activities:

- Determining test cases
- Entering test data and expected values
- Executing the test
- Evaluating and documenting the test

Usually you would import your requirements first. To keep this exercise understandable for beginners, we will first exercise a simple project, then import some basic requirements and restart the test!

We will now follow a simple source code example to show how to excercise those activities with TESSY.

Example "is_value_in_range"

A start value and a length define a range of values.
Function: Determine if a given value is within the defined range or not. Only integer numbers are to be considered.

5.1.1 Creating a test project

If you have not created the project "Example1" yet, do as follows:

To understand TESSY´s file system and databases, consult section 3.1 Creating databases and working with the file system.

→ Start TESSY by selecting "All Programs" > "TESSY" > "TESSY 3.0".

→ If the "Open Project" dialog will open, click "New Project".

If another project is opened within TESSY, click "File" > "Select Project" > "New Project".

→ Enter ***Example1***.

→ Select the path for the "Project Root" of your project: Create a folder "MyProjects" and within a folder "TESSYProjectABC".

The "Database Root Path" will be defined automatically (see figure 5.2).

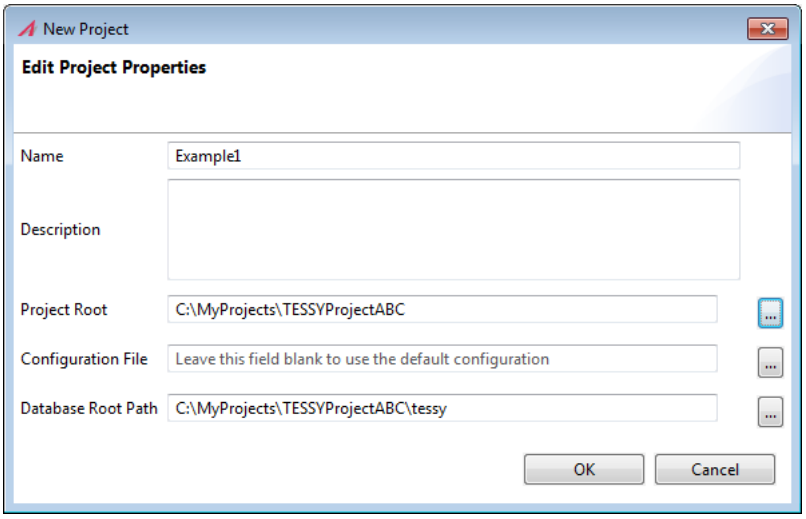


Figure 5.2: Creating the new project "Example1".

→ Click "OK". TESSY now creates the project "Example1". This will take a few seconds.

→ Click "Open" when TESSY has finished.

TESSY now opens your project. This will take a few seconds.

Organizing "Example1"

The project "Example1" is opened within the Overview perspective. You can create different folders within a test collection, each containing modules with various test objects. To keep it simple, we will create now one test collection with one folder.

We start within the view "Test Project":

→ In the Test Project view click on the icon (New Test Collection) in the tool bar of the view.

→ Enter ***Is_value_in_range*** and press the "Enter"-key.

→ Click on (New Folder), enter ***ExampleFolder***, click "Enter".

→ Click on (New Module), enter ***ExampleModule***, click "Enter".

The module relates to one or many source files which are to be tested.

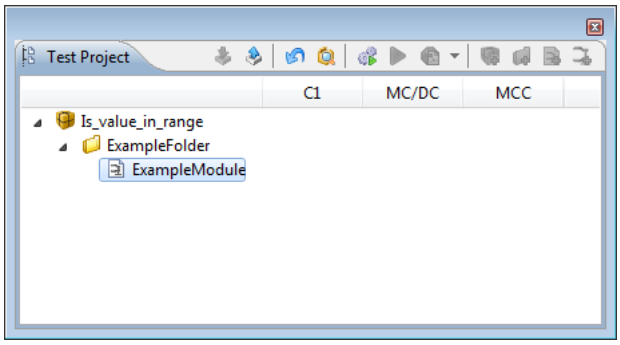


Figure 5.3: Test collection "Is_value_in_range" with an example folder and module

Rename or delete a module or a folder by using the context menu (right click > "rename" or "delete") or the key F2.

5.1.2 Specifying the target environment

Usually at this point you will have to specify the target environment, that is to determine the compiler, the target and the microcontroller. You will do that in the "Test Environment Editor" which we will get to know later.

Please notice beneath in the Properties view at tab "General" that the GNU GCC compiler is already selected for this module (see figure 5.4):

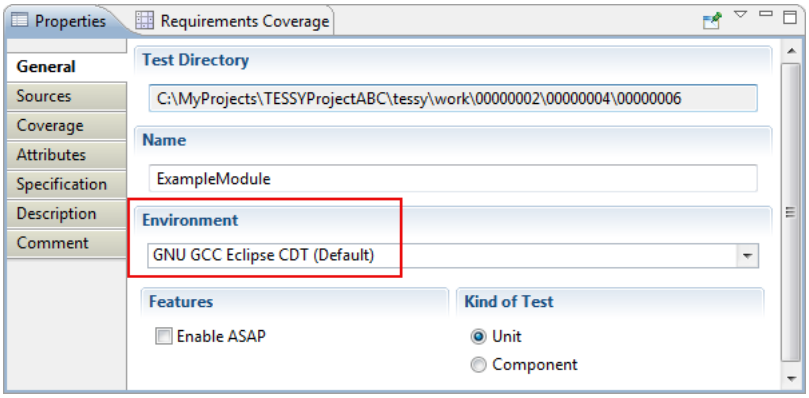


Figure 5.4: GNU GCC compiler is selected by default.

C-source file

The GNU GCC compiler is included with the TESSY installation and enough for our example. Now we will add the source file to the module. The source file contains the C-function to be tested:

```
01 // This is a very basic program to demonstrate the integration
02 // of Tessy and HiTOP, the debugger from Hitex.
03 // (c) Hitex Systementwicklung GmbH 2001, www.hitex.de
04 //
05 // $Revision: 2$
06
07 struct range {int range_start; int range_len;};
08
09 typedef int value;
10
11 typedef enum {no, yes} result;
12
13 // Checks if r1.start <= v1 < (r1.start + r1.len),
14 // e.g. r1.start = 5, r1.len = 2
15 // ==> v1 == 4 ---> no
16 //     v1 == 5 ---> yes
17 //     v1 == 6 ---> yes
18 //     v1 == 7 ---> no
19 // However, the implementation is intentionally
20 // erroneous: v1 == 7 results "yes" instead of "no"!
21 //
22
23 result is_value_in_range (struct range r1, value v1)
24 {
25     if (v1 < r1.range_start)
26         return no;
27
28     if (v1 > (r1.range_start + r1.range_len))
29         return no;
30
31     return yes;
32
33 }
```

Figure 5.5: The source code of the C-Function to be tested

5.1.3 Adding the test object and analyzing the C-source file

We will use the example C-source file "`is_val_in_range.c`" which is stored under "`C:\Program files\Razorcat\tessy_3.0\examples\IsValueInRange`". Copy the C-source file, paste it in the project root an add it to the module:

→ Open the Windows Explorer.

→ Copy the source file "`is_val_in_range.c`" in a folder which is located in the project root, e.g. "`c:\MyProjects\TESSYProjectABC\sources`".

It is useful to relate all sources, includes etc. to the project root. You have a better overview about all sources, includes etc.

→ Switch back to TESSY.

→ In the Test Project view select the module ("Example Module").

→ In the Properties view switch to tab "Sources".

→ Click on (Add Source).

Adding the C-source

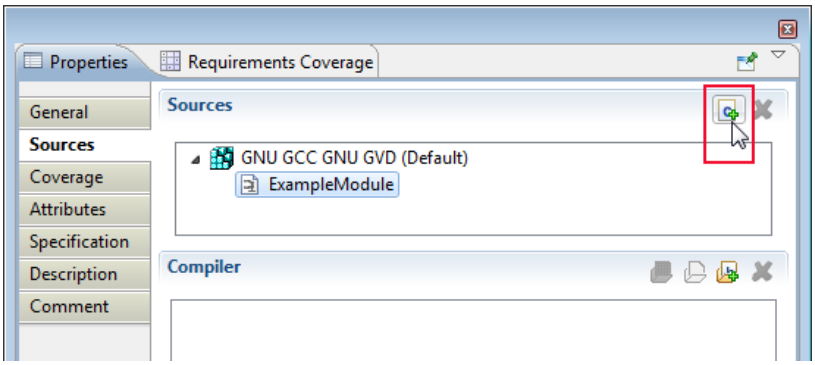


Figure 5.6: Adding the C-source file.

→ Select the source file "`is_val_in_range.c`" from the folder where you just pasted the source.

→ Click "Open". The C-source file will be added.

→ In the Test Project view above click on (Analyze Module) to start the module analysis (see figure 5.7).

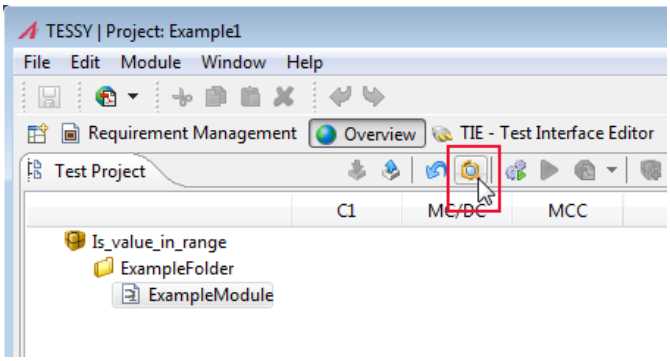
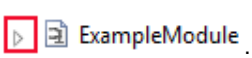


Figure 5.7: Analyzing the module, that is the C-source file.

TESSY now analyzes the C-source file, this will take a few seconds. After successful processing,

→ click on the white arrow in front the module:  .

TESSY will as well analyze the C-source file by just clicking on the white arrow after adding the C-source-file.

Now all functions which were defined in the C-source file are displayed as children of the module above within the Test Project view (see figure 5.8).

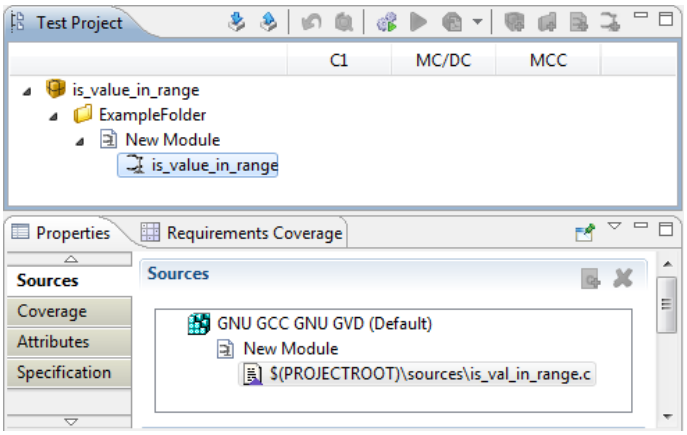


Figure 5.8: The function of the C-source is displayed as child of the module.

Our sample C-source file contains only one function, our **test object** "is_value_in_range".

The term "test object" indicates the functions within the module we are attempting to test.

5.1.4 Editing the test object interface

→ Switch to the TIE (Test Interface Editor).

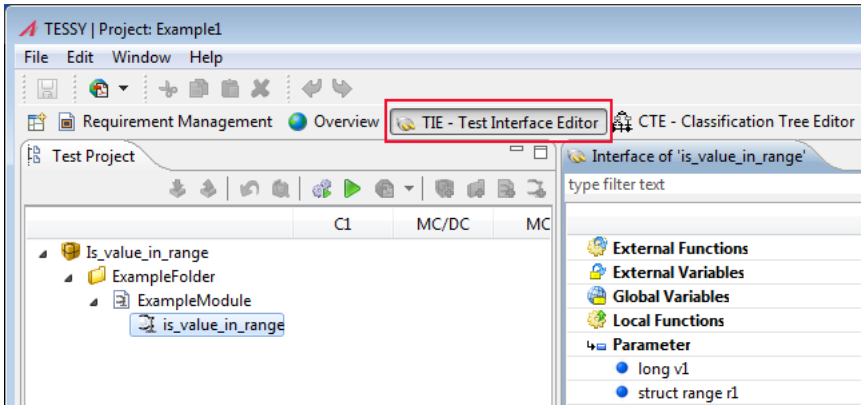


Figure 5.9: Perspective TIE - Test Interface Editor

Determine passing directions

Now we can edit the interface information for every test object and determine which values are input and which ones are output variables. **Input values** are all interface elements that are read by the test object. **Output values** are written by the test object.

Upon opening the module, TESSY will try to set the default passing directions (input or putput) automatically. You can change these default interface settings to your needs.

In our sample the passing directions are already defined, you do not have to take actions.

→ In the Interface view open the Parameter paragraph to see the inputs and output values that are already defined in our example (see figure 5.10)

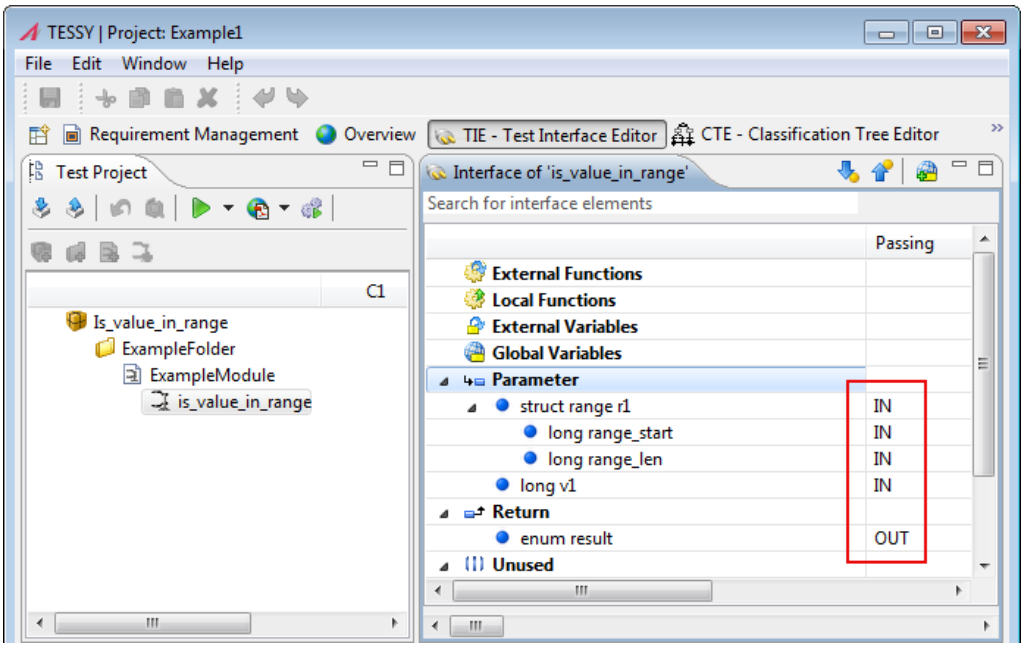


Figure 5.10: The inputs and outputs are already defined

5.1.5 Designing test cases

Usually now you would design the test cases, either manually or within the Classification Tree Editor CTE, based on specifications of your test object.

Since the Classification Tree Editor (CTE) is a subject for its own, we will not make use of the CTE in this example, but simply enter some ad-hoc test data manually.

To learn about the CTE refer to section 6.6 CTE: Designing the test cases or follow the Quickstart 2: Exercise Classification Tree Editor (CTE).

5.1.6 Adding test cases and test steps

Now we will add three test cases each with one test step within the Test Items view:

→ Switch to Overview perspective and to view Test Items.

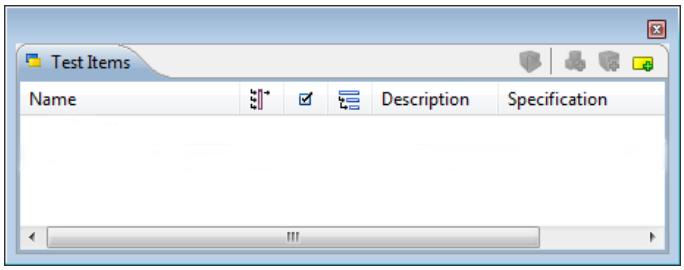


Figure 5.11: Test Items view

→ Select the test object in the Test Project view.

→ In the Test Items view click on (New Test Case).

The first test case is created and a test step is automatically added.

In TESSY every test case has at least one test step.

→ Add two further test cases.

→ Expand the test cases by clicking on the arrows in front of the test cases.

Adding test cases

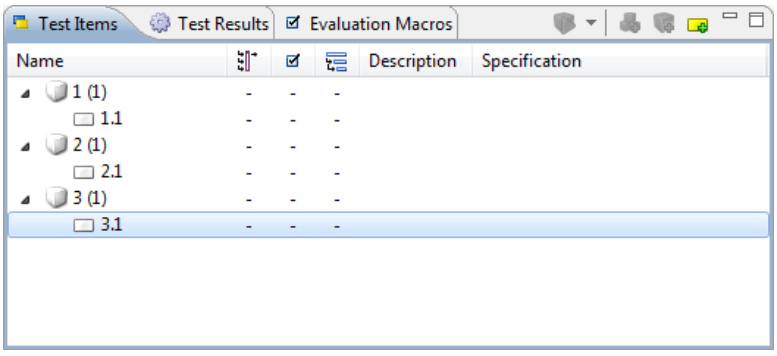


Figure 5.12: Three test cases were added in the Test Items view

Please notice the following habits of this view:

- The first number is the number of the test case, the number in brackets shows the quantity of the test steps included.
- Test case numbers will be counted continuously: If you delete test cases, new test cases will get a new number and existing test cases will not be renumbered.
- If you cannot click on "New Test Case" oder "New Test Step" because the icons are inactive, you might be in the wrong selection: Select the test object within the Test Project view, then select the Test Items view.

5.1.7 Entering test data

Now we will define some input and output values:

→ Switch to the perspective "TDE - Test Data Editor". The TDE will also open with a double click on a test case or a test step.

In the Test Data view you can see the test cases and steps in tabular form.

→ Under "Inputs" click on the arrow to open "struct range r1".

→ For test case 1 (1.1) enter *3* for "range_start".

→ Enter *2* for "range_len".

→ Enter *4* for "v1".

→ Click on (Save) to save your inputs.

After saving, the symbol of the test object in the Test Project view as well as the symbol of the test case in the Test Items view turns yellow to indicate that the test case is ready to run (see figure 5.13).

→ Under "Outputs" click on the arrow ahead "Return".

→ Enter "yes" for the return value.

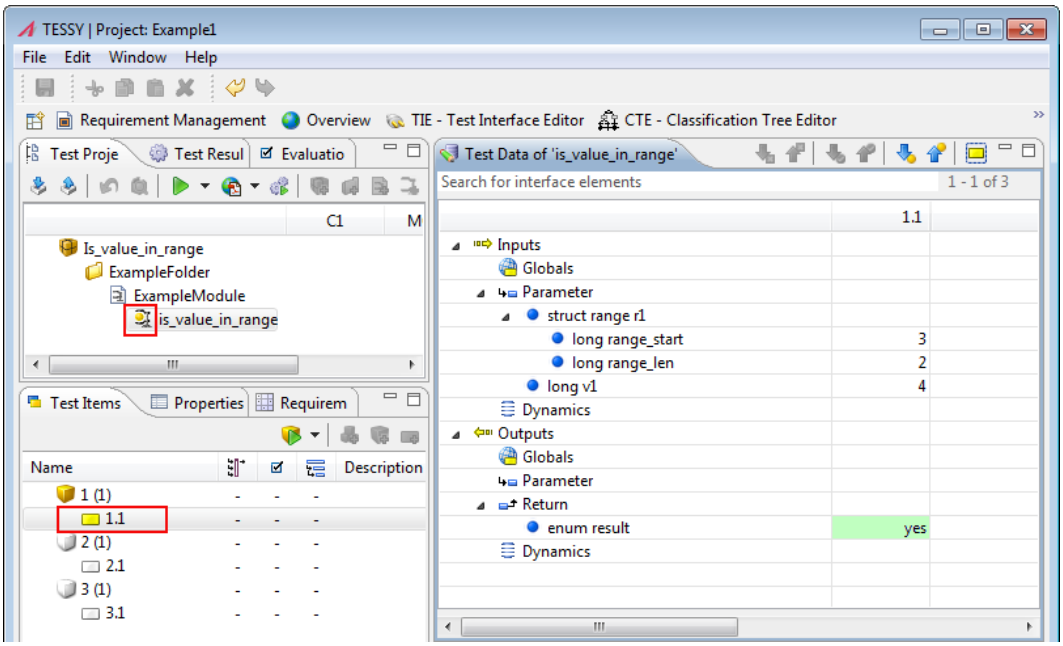


Figure 5.13: Data is entered, test step turns yellow and test case is ready to run.

Please notice the changes of the test object icon to indicate different conditions:

- Empty gripper: Test object is analyzed but has no test case.
- Gripper with white object: Test object has test cases but no data.
- Gripper with yellow object: At least one test step is ready to be executed.

→ Now enter data for the other two test cases as shown in table 5.17.

test step:	1.1	2.1	3.1
range_start:	3	20	0
range_length:	2	8	5
v1:	4	22	6
expected return:	yes	no	no

Table 5.17: Entering data for test object is_value_in_range.

- Test case 1.1: The range starts at **3** and has a length of **2**. Therefore, the range ends at 5 and the given value 4 is supposed to be inside of the range (**yes**).
- Test case 2.1: The range starts at **20** and has a length of **8**. Therefore, the range ends at 28 and the given value 22 is supposed to be inside of the range. Because we want to force an incorrect output, we state this to be not inside of the range (**no**).
- Test case 3.1: The range starts at **0** and has a length of **5**. Therefore, the range ends at 5 and the given value 6 is supposed **NOT** to be inside of the range (**no**).

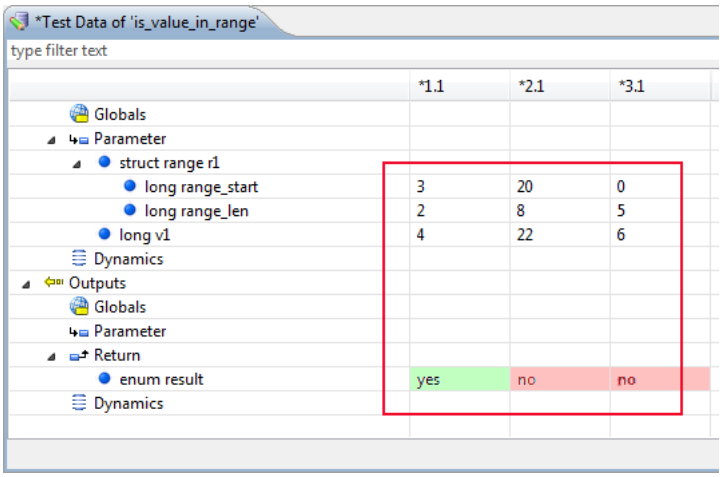


Figure 5.14: Entering data for test object is_value_in_range

The test step icons in the Test Items view will now turn to yellow (see figure 5.15). This indicates that we are now ready to run the test.

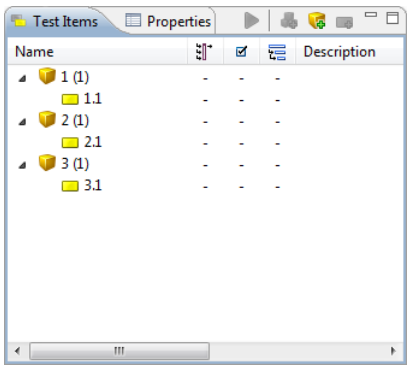


Figure 5.15: The test cases are ready to test

5.1.8 Executing the test

→ Click on (Start Test Execution) in the tool bar of the Test Project view.

A progress dialog will be shown while TESSY generates, compiles and links the test driver and runs the test. This will take a few seconds.

After the test run, test case icons (within TDE) should be (see figure 5.16):

- The expected result "no" of test case 2 is marked red to indicate that this does not match the actual result.
- The test collection as well as the folder and module and test object are marked with a red cross to indicate, that the results did not match the ecpected results.
- The Test Items view indicates with green checks, that test case 1 and 3 did match the expected results.
- The Test Items view indicates with a red cross, that test case 2 did not the expected result.

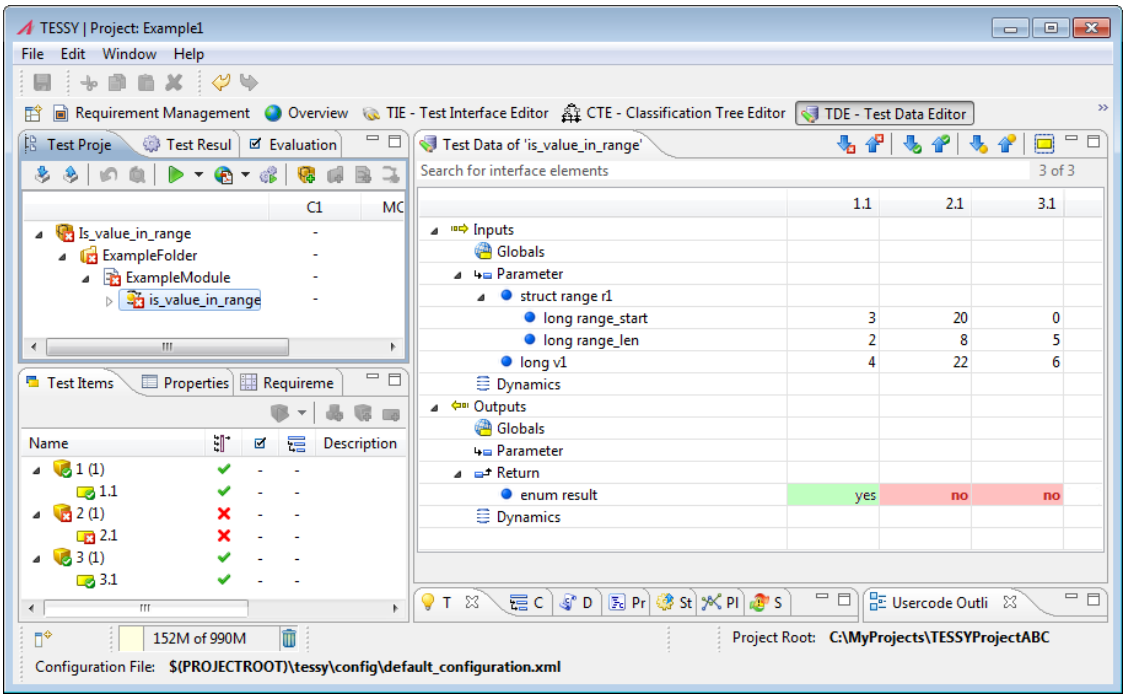


Figure 5.16: TDE after test run is_value_in_range

→ Switch to the Overview perspective.

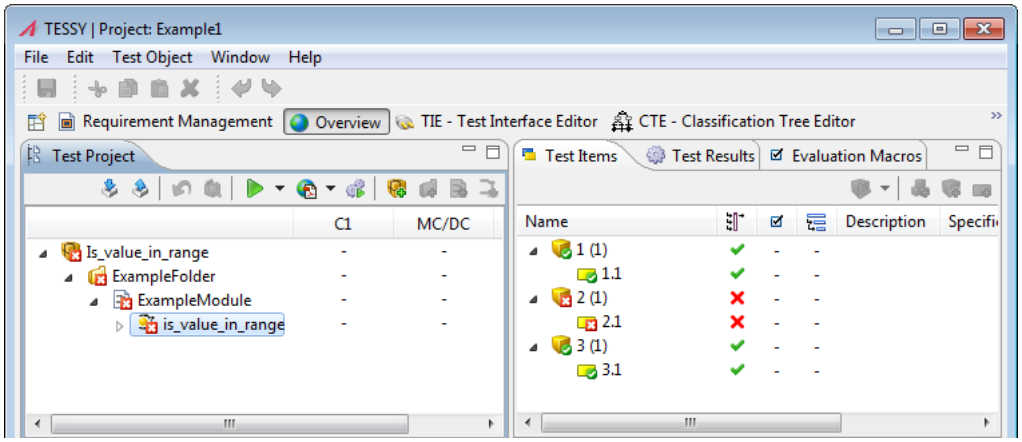


Figure 5.17: Test results of is_value_in_range

You can see the results of every test step within the Test Results view.

5.1.9 Repeating the test run with coverage instrumentation

To analyze the source code coverage of the test, repeat the test run with the branch, MC/DC and MCC-coverage instrumentation:

→ In the tool bar of the Test Project view click on the arrow next to the Execute Test icon and select "Edit Test Execution Settings . . . ".

→ In the following dialog choose "Run" (default), "Create New Test Run" and the instrumentation "Branch and MC/DC, MCC Coverage - Test Object" (see figure 5.18).

→ Click on "Execute".

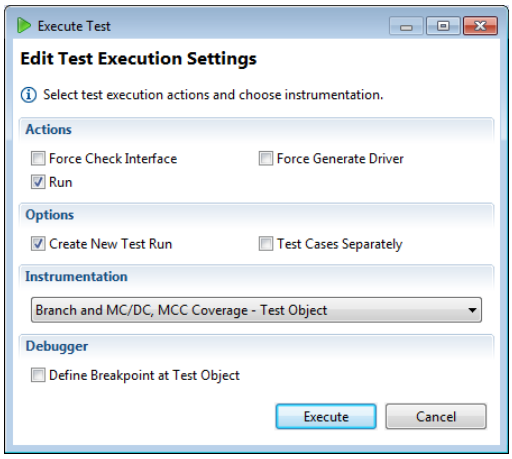


Figure 5.18: Selecting Branch and MC/DC Coverage for test run

A progress dialog will be shown while TESSY generates, compiles and links the test driver and runs the test. This will take a few seconds.

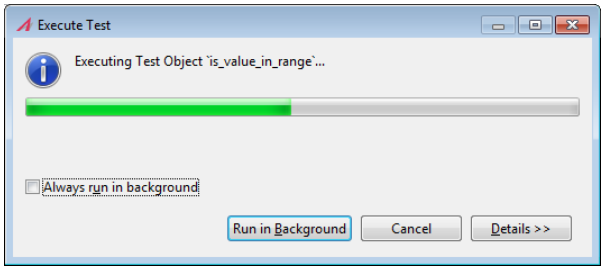


Figure 5.19: Execute Test dialog while running the test

5.1.10 Analyzing the coverage

→ Switch to the Coverage Viewer (CV) perspective.

Analyzing with the CV

The CV shows the results of the coverage measurement of a previously executed test. TESSY provides the following coverage measurements:

- C1 branch coverage
- MC/DC modified condition
- MCC multiple condition coverage

The flow chart view

The flow chart view displays the code structure and the respective coverage in graphical form. Within each flow chart, you will see the decisions and branches of the function being displayed. Green and red colors indicate whether a decision has been fully covered or a branch has been reached.

The Branch (C1) Coverage view

The Branch C1 Coverage view displays the branch coverage for each individual test case and test step as well as the total coverage for all test cases and test steps.

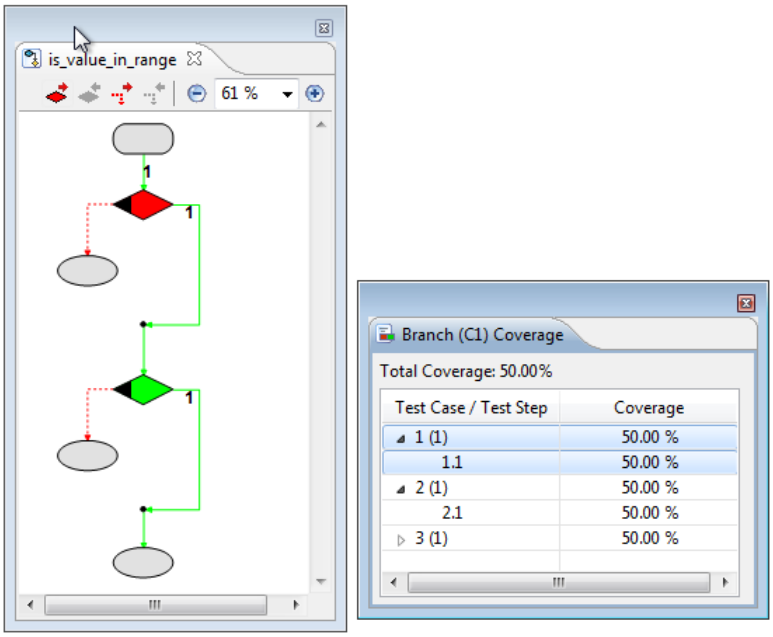


Figure 5.20: Branch coverage is_value_in_range

The MC/DC Coverage view

The MC/DC-Coverage view displays the coverage of the currently selected decision within the flow chart view (see figure 5.21). If no decision is selected (as initially when starting the CV), the MC/DC Coverage view is empty.

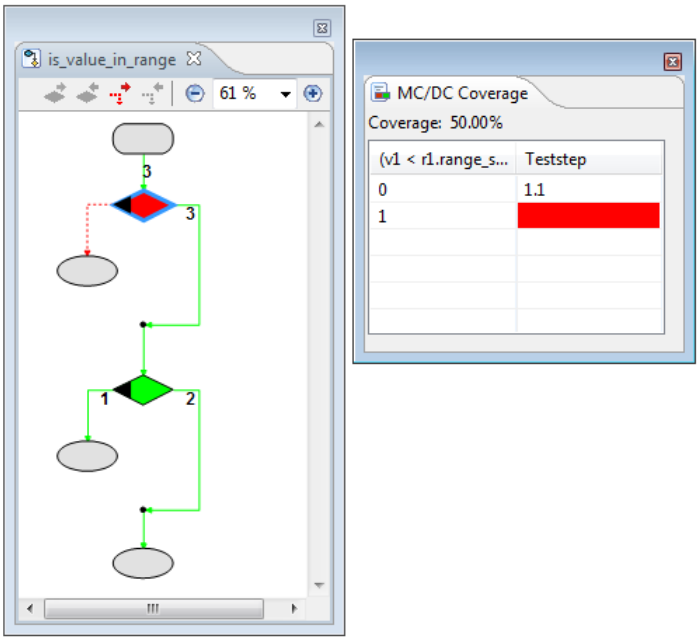


Figure 5.21: Decision coverage is_value_in_range

The current example is_value_in_range has only simple decisions, for which MC/DC is the same as branch coverage.

Analyzing

In our example you can see in the flowchart, that

- three test cases were executed (each with with one test step).
- the first decision has an else branch on the right, that was executed three times.
- the if branch on the left of the first decision was not reached and is therefore marked red.
- the first decision was not fully covered, so it is marked red.
- the second decision was fully covered and is therefore marked green.
- the else branch on the right of the second decision was reached two times, the else branch was reached once.

→ Select the red branch of the first decision (the if branch).

The respective code section is highlighted within the source code view (see figure 5.22). This allows finding out the execution path of the selected test step.

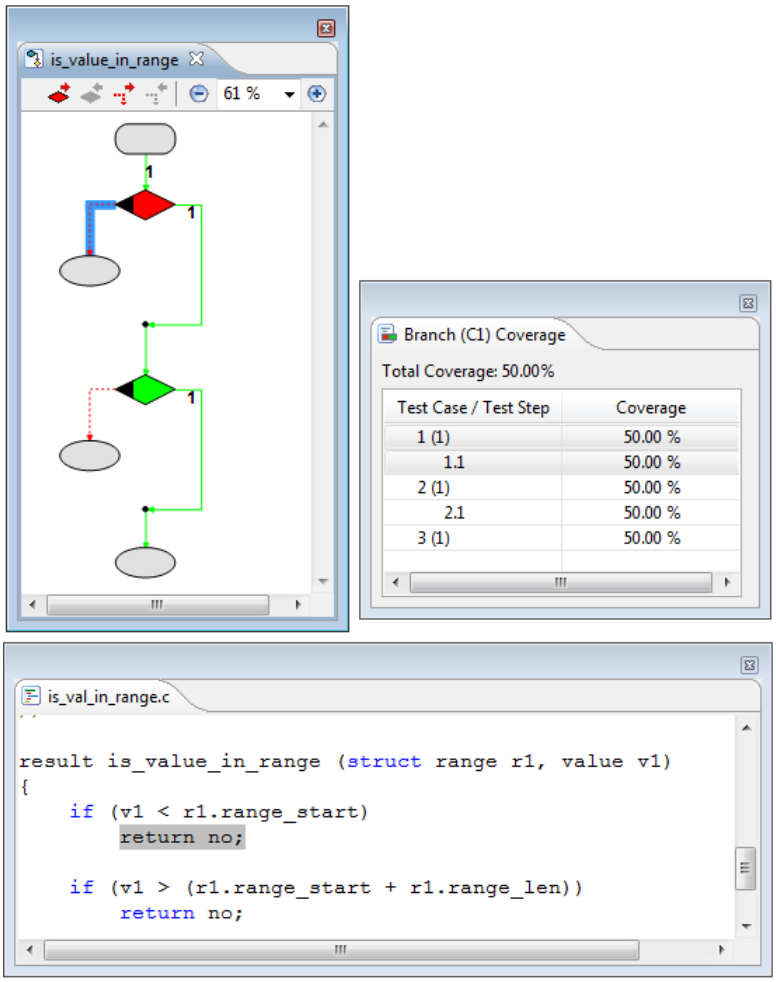


Figure 5.22: Code section of the if branch of the first decision

→ Select the second decision.

The respective code section is highlighted within the source code view (see figure 5.23).

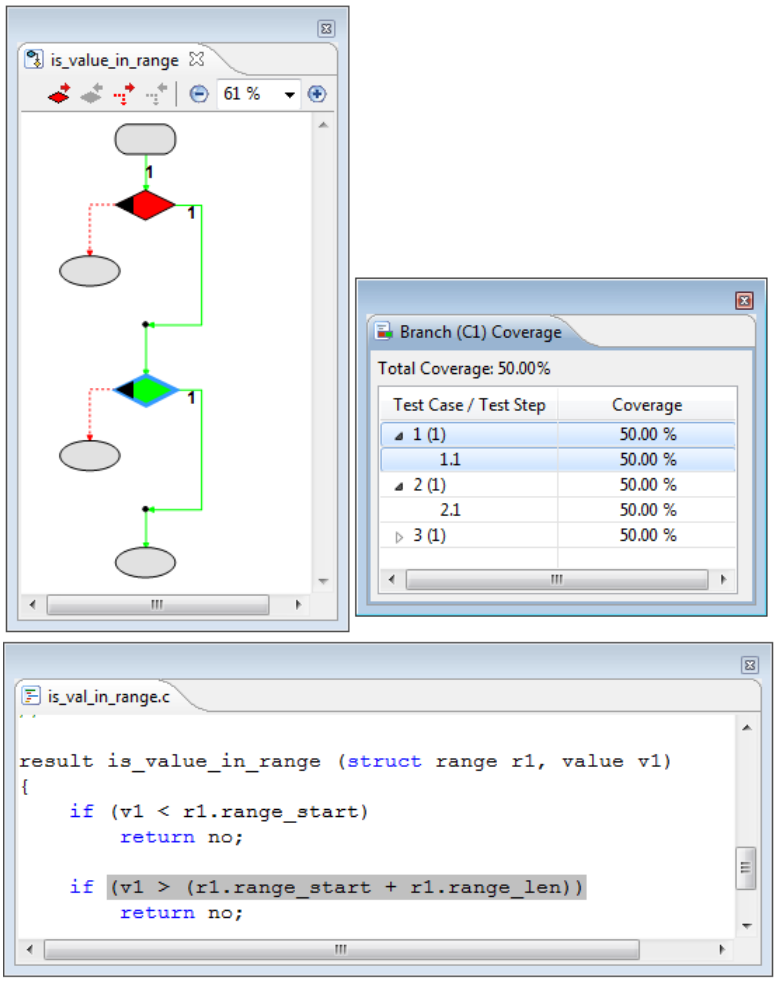


Figure 5.23: Code section of the second decision

5.1.11 Creating a report

→ In the Test Project view of the Overview perspective click on the arrow next to the Generate Report icon and select "Edit Test Details Report Settings...".

→ In the dialog click on the button "Browse..." and select your project folder "'Example1".

→ Click on "Make New Folder" and create a folder "Reports" (see figure 5.24).

→ Click on "OK" and "Generate".

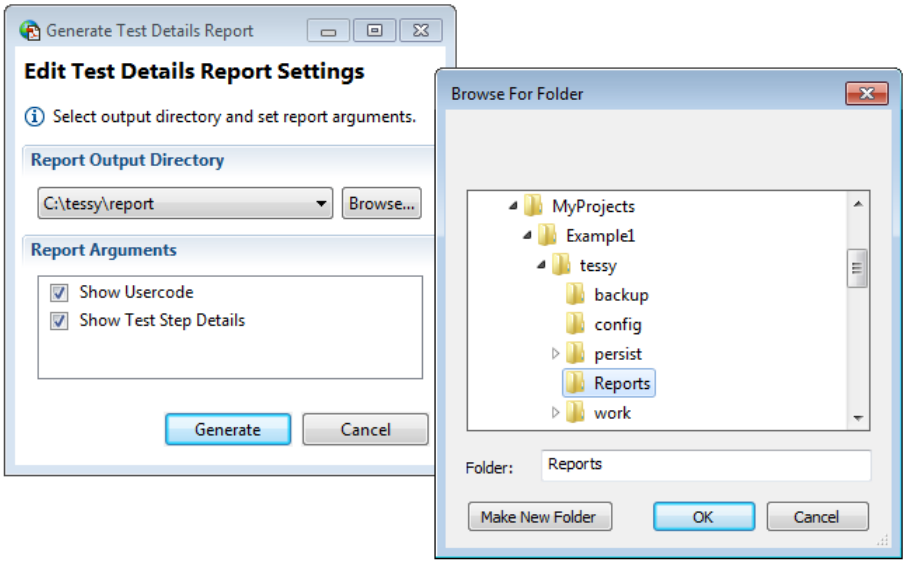


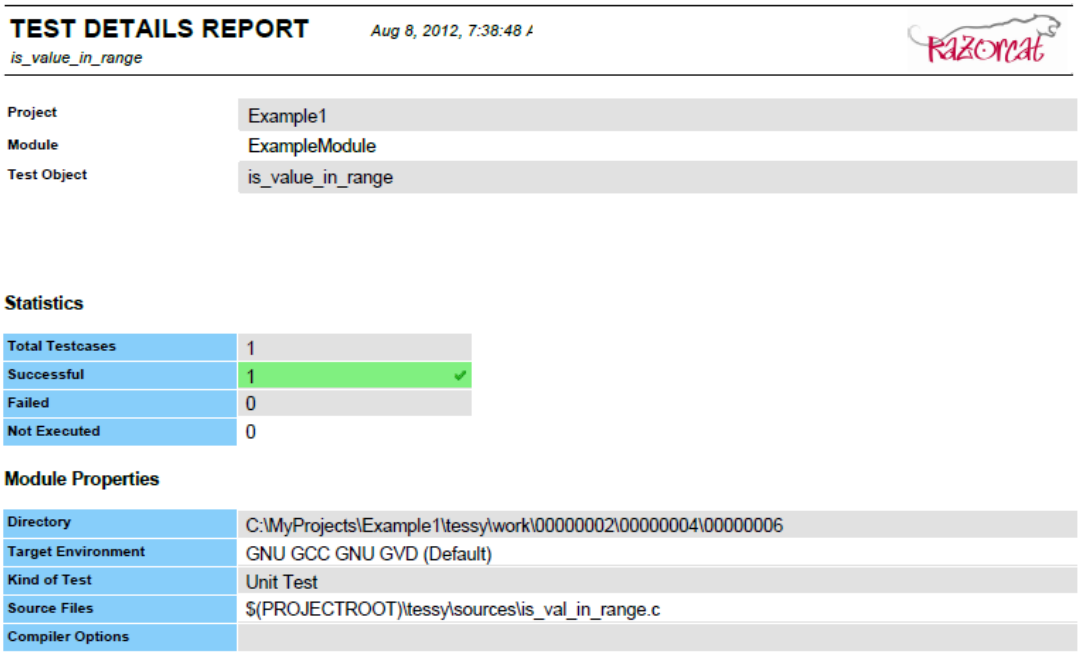
Figure 5.24: Creating the folder for reports

TESSY creates the report within the new folder. This will take a few seconds.

The selection of the folder is memorized and used for the following.

When finished, TESSY will open the file automatically.

Important: If you get an error "No matching program found for the file ...", then the Adobe Reader is not installed, which you need to open the reports (PDF files). Download and install the Adobe Reader from http://get.adobe.com/reader/ and generate the report again.



TEST DETAILS REPORT Aug 8, 2012, 7:38:48 A

is_value_in_range

Project	Example1
Module	ExampleModule
Test Object	is_value_in_range

Statistics

Total Testcases	1
Successful	1
Failed	0
Not Executed	0

Module Properties

Directory	C:\MyProjects\Example1\tessy\work\00000002\00000004\00000006
Target Environment	GNU GCC GNU GVD (Default)
Kind of Test	Unit Test
Source Files	$(PROJECTROOT)\tessy\sources\is_val_in_range.c
Compiler Options	

Figure 5.25: Content of the test report is_value_in_range

Now you have accomplished your first test project!

You can easily change the default Razorcat logo within the reports to your own company logo: In the menu bar select "Window" > "Preferences". Select the General Settings page > "Test Report" > "LogoImage". Enter the path to your logo image file (PNG, JPG or GIF format).

5.1.12 Repeating the test run with requirements

We will now import some very basic requirements and repeat some steps of this exercise. This way you get to know the feature of requirement management and can consolidate the just learned workflows.

Requirement Management

Importing requirements

→ Copy the example requirements document "Is Value In Range Requirements.txt" into a folder of your test project "TESSYProjectABC". It is located within the TESSY installation directory `\Razorcat\TESSY_3.0\Examples\IsValueInRange`.

→ Switch to the Requirement Management perspective.

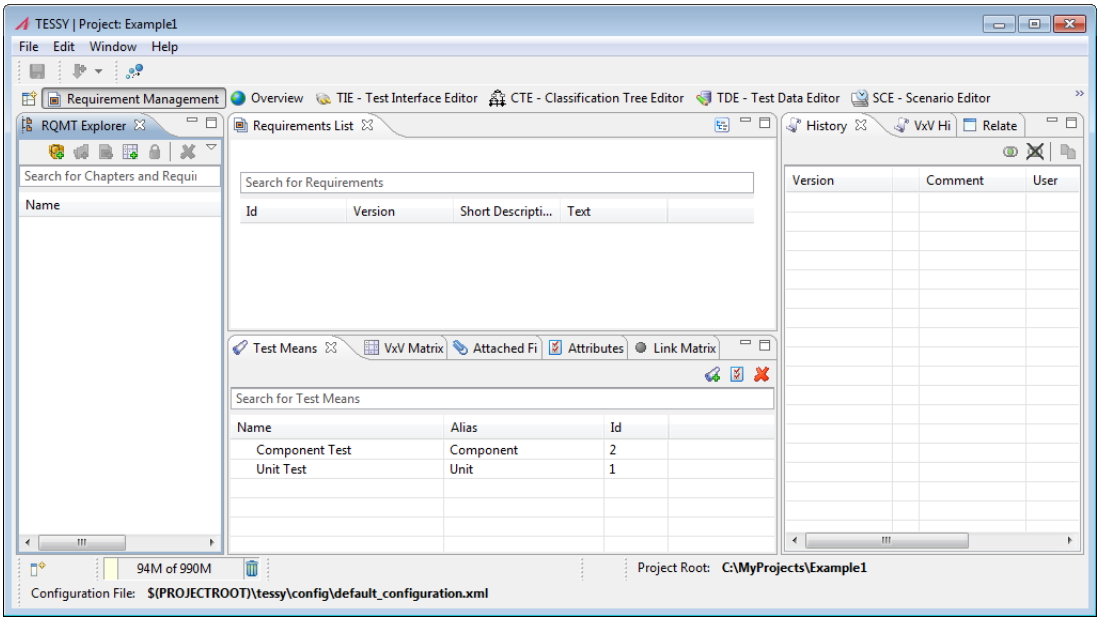


Figure 5.26: Requirement Management perspective

→ In the global tool bar click on (Show Import Perspective).

The Import perspective will open.

The Import perspective provides the means to import various file formats (see figure 5.27).

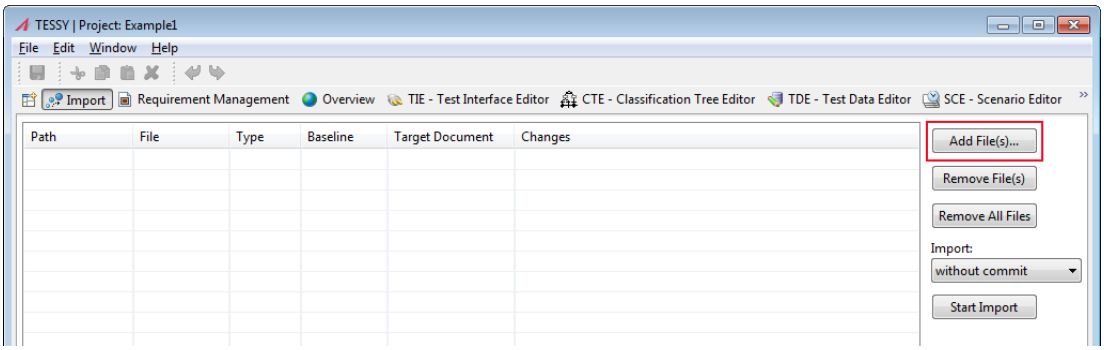


Figure 5.27: Import perspective of the requirement management

→ Click on "Add File(s)..." and select the file "Is Value In Range Requirements.txt".

The file will be displayed within the table.

→ Leave the proposed name as is and click the button "Start Import".

The Import Log dialog shows the status of the import operation.

→ Switch back to the Requirement Management perspective.

The newly imported requirement document will be displayed in the RQMT Explorer view (see figure 5.28).

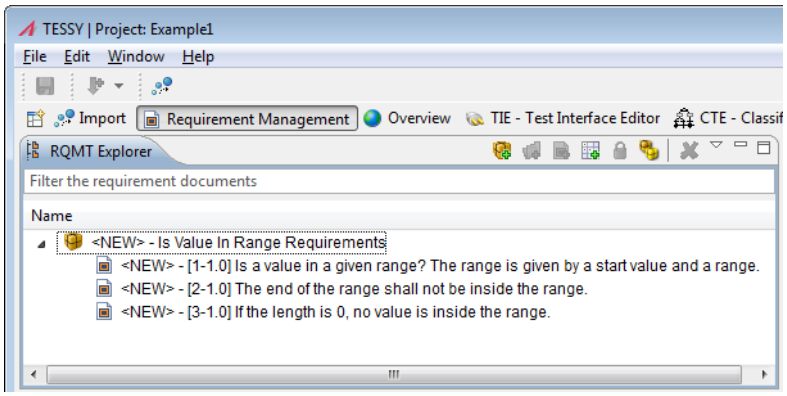

TESSY | Project: Example1
File Edit Window Help
Import
Requirement Management
Overview
TIE - Test Interface Editor
RQMT Explorer
Filter the requirement documents
Name
<NEW> - Is Value In Range Requirements
<NEW> - [1-1.0] Is a value in a given range? The range is given by a start value and a range.
<NEW> - [2-1.0] The end of the range shall not be inside the range.
<NEW> - [3-1.0] If the length is 0, no value is inside the range.

Figure 5.28: The new requirement document

→ Select the document and select "Rename" from the context menu.

→ Enter ***ivir_requirements*** as name.

→ Enter ***IVIR*** as alias.

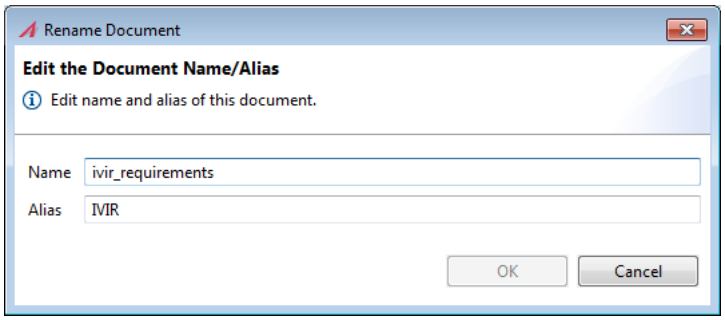


Figure 5.29: Entering an alias for the requirement document

The document alias will be used for reporting, in order to have an abbreviation of the document name when building the requirement identifier, e.g. IVIR-[1.0] in our example.

Committing the requirements document

Committing

Before linking any tests to a requirement, the respective requirements document needs to be checked in as initial revision:

→ Select the document and click on (Commit Changes) in the global tool bar.

→ Enter "Initial revision" as commit comment and click "OK" (see figure 5.30).

An initial revision of the requirement document will be created.

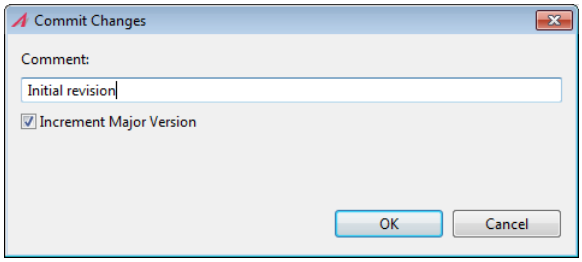


Figure 5.30: Comment for the initial revision of the commit

TESSY manages different versions of a requirements document. You can track any changes either from importing updated versions or from any modifications that you did directly within TESSY.

Linking test cases with requirements

→ Switch to the Overview perspective and select the test object.

→ Select the Requirements Coverage view.

→ Click on (Always show unlinked Requirements).

The view shows the imported requirements and the module, test object and test cases in a tree-based arrangement.

> **Important:** If the view says "No requirements available, select the test object in the Test Project view!

Use the toggle buttons on the right to link modules, test objects or test cases to requirements:

→ Link the first test case with the first requirement.

→ Link the second test case with the second requirement (see figure 5.31).

The third requirement is not linked.

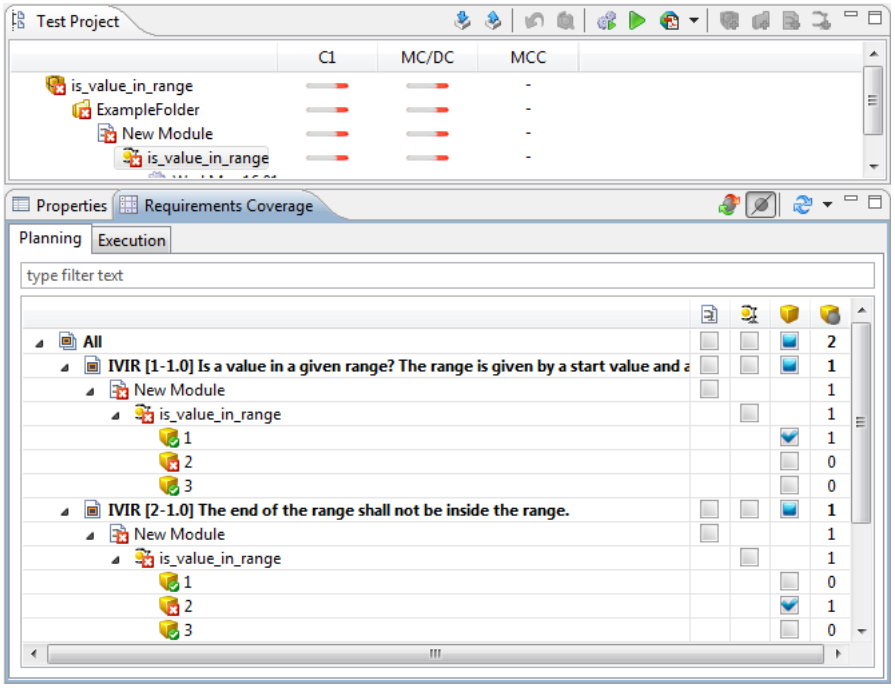


Figure 5.31: Linking test cases with requirements

Remember the values of our test cases:

- Test case 1.1: range start 3, length 2, given value 4, supposed to be inside of the range (yes)
- Test case 2.1: range start 20, length 8, given value 22. Because we wanted to force an incorrect output, we stated this to be not inside of the range (no).

→ Switch to the TDE perspective.

→ In the Test Item view select the first test case and have a look at the Test Definition view: It shows the requirements we just linked with our test cases.

→ Select the second test case.

The second requirement will be displayed (see figure 5.32).
Since we did not link any requirement to the third test case, the "Linked Requirements" will be empty when selecting the third test case.

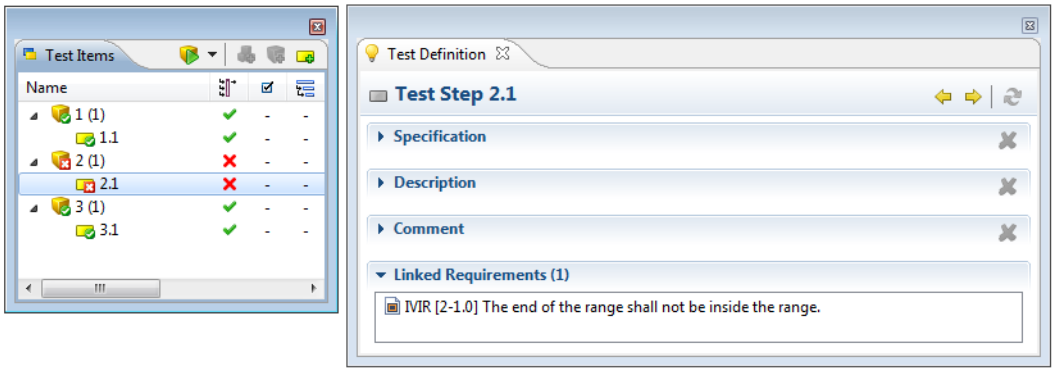


Figure 5.32: Test Definition view within TDE with linked requirement

Creating a planning coverage report

At this stage, we can already generate a report showing the planned test case for our requirements:

→ Switch to the Test Project view of the Overview perspective and click on the arrow next to the Generate Report icon .

→ Select "Edit Planning Coverage Report Settings..." (see figure 5.33).

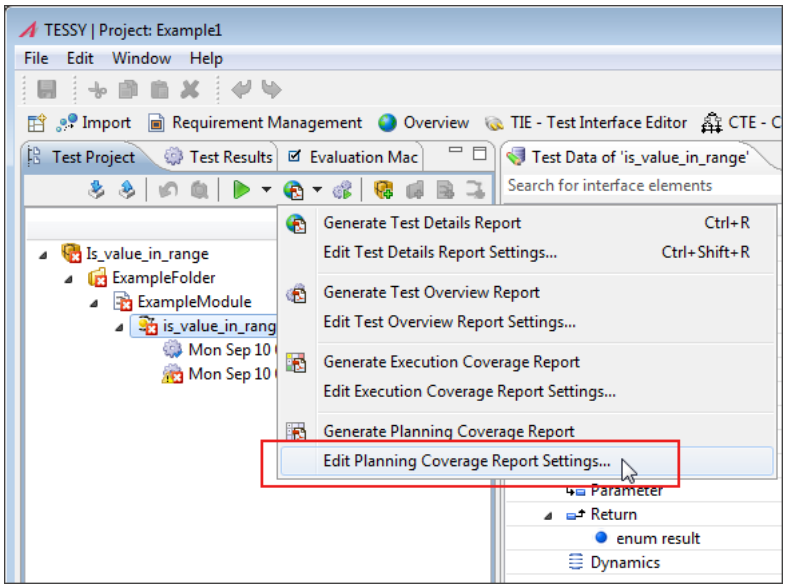


Figure 5.33: Editing the settings of a Planning Coverage Report

→ A dialog for the settings for the Planning Coverage Report will open.

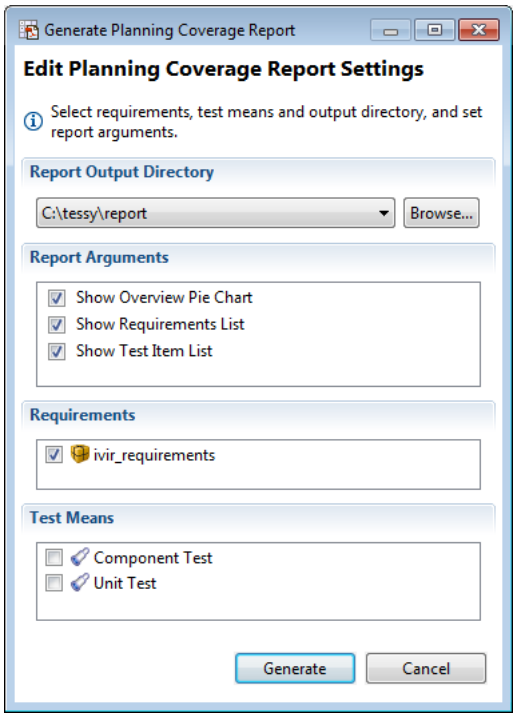


Figure 5.34: Dialog of the settings for the Planning Coverage Report

→ Select an output directory for the report (default: `C:\tessy\report`).

→ Select all Report Arguments (default).

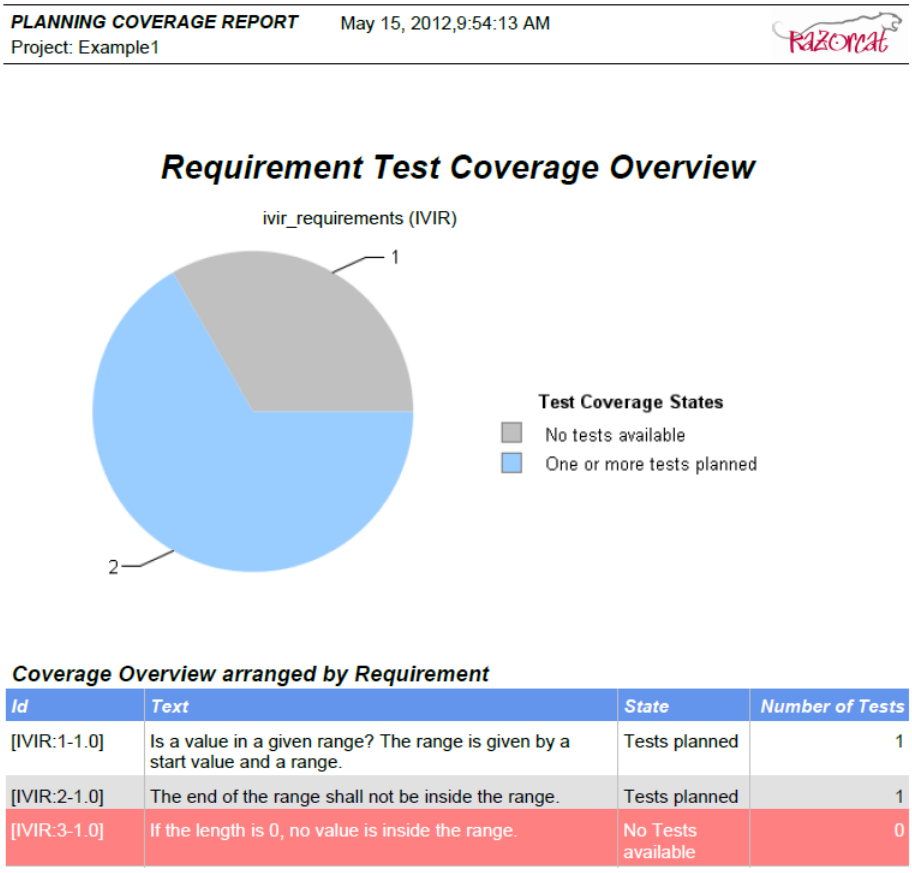
→ Select the ivir_requirements.

→ Do NOT select any test means.

→ Click on "Generate".

A planning coverage report will be created.

Planning Coverage Report



PLANNING COVERAGE REPORT May 15, 2012,9:54:13 AM
Project: Example1

Requirement Test Coverage Overview

ivir_requirements (IVIR)

Coverage Overview arranged by Requirement

Id	Text	State	Number of Tests
[IVIR:1-1.0]	Is a value in a given range? The range is given by a start value and a range.	Tests planned	1
[IVIR:2-1.0]	The end of the range shall not be inside the range.	Tests planned	1
[IVIR:3-1.0]	If the length is 0, no value is inside the range.	No Tests available	0

Figure 5.35: Planning coverage report of the IVIR requirement document

The report shows the available requirements and the linked test cases. It provides an overview about the planned tests and if all requirements are covered by at least one test case.
Since we have links to two of our requirements, the resulting requirement coverage should be as shown above.

Notice the usage of the requirement document name and alias within the report! It is important to select an appropriate alias in order to get useful report outputs.

We have planned test cases for the first two requirements, whereas the third requirement is not yet linked with any test case, because there are no tests available to validate this requirement.

Executing the test and examining the coverage

We will now execute our tests again to see the results of the test cases with respect to the linked requirements within the execution coverage report.

→ Switch to the Overview perspective and execute our test object is_value_in_range again: Click on , choose none instrumentation and click "OK".

→ Generate a test details report to review the results on test object level (see figure 5.36).

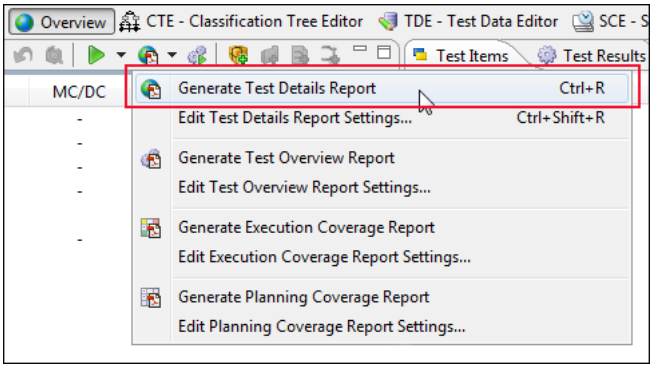
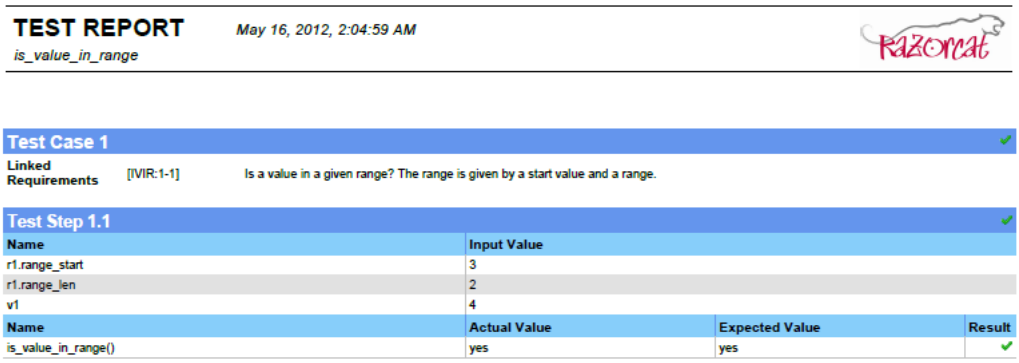


Figure 5.36: Generating a Test Details Report

The report will open automatically.

→ Have a look at the second page.

The report will show additional paragraphs with the linked requirements after the overview pages and for each test case (see figure 5.37):



TEST REPORT *May 16, 2012, 2:04:59 AM*
is_value_in_range

Test Case 1

Linked Requirements	[IVIR:1-1]	Is a value in a given range? The range is given by a start value and a range.

Test Step 1.1

Name	Input Value
r1.range_start	3
r1.range_len	2
v1	4

Name	Actual Value	Expected Value	Result
is_value_in_range()	yes	yes	✔

Figure 5.37: Part of the generated test report of is_value_in_range

Now we will generate a coverage report showing the test case results with respect to our requirements:

Test Object Report

→ In the global tool bar click on the arrow next to the Generate Report icon and select "Generate Execution Coverage Report" (see figure 5.38).

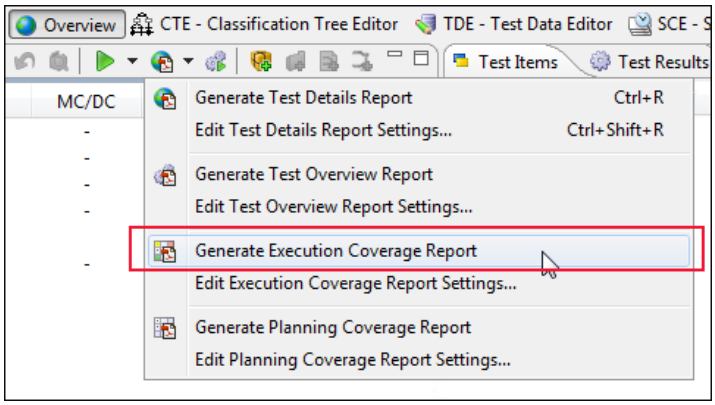


Figure 5.38: Creating an Execution Coverage Report

The settings dialog will open.

→ Choose the ivir_requirements document and click on "Generate".

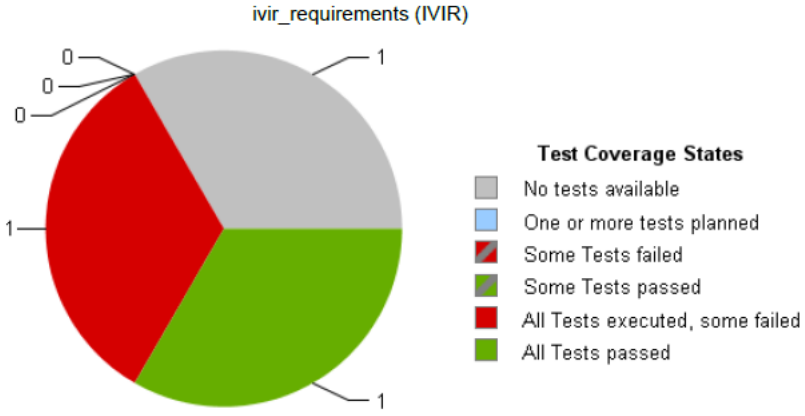
TESSY creates the coverage report, showing the available requirements and the results of the linked test cases. It provides an overview about the current test status, e.g. if tests for any requirements are failed.

Since one of our test cases was passed while the other one was failed, the resulting requirement coverage should be as in figure 5.39.

Execution
Coverage Report

TEST COVERAGE REPORT
Project: Example1

Requirement Test Coverage Overview



Coverage Overview arranged by Requirement

Id	Text	State	Number of Tests
[IVIR:1-1.0]	Is a value in a given range? The range is given by a start value and a range.	All Tests passed	1
[IVIR:2-1.0]	The end of the range shall not be inside the range.	All Tests executed, some failed	1
[IVIR:3-1.0]	If the length is 0, no value is inside the range.	No Tests available	0

Figure 5.39: Coverage Report of is_value_in_range

The first requirement has one test case linked which was successfully executed, the second requirement has also one test case linked, but this one failed. The third requirement has still no test case assigned.

Now you finished successfully the exercise is_value_in_range.

5.1.13 Reusing a test object with a changed interface

If the interface of the test object changes, TESSY will indicate the changes with specific test readiness states. With the Interface Data Assigner (IDA) you can assign the elements of a changed (new) interface to the elements of the old one.

Using IDA

In this section we will change the interface of the test object by editing the C-source and exercise a reuse operation within the IDA.

Important: Make sure to keep the original C-source file "is_val_in_range.c" and edit a copy. Do not change the original file in folder "C:\Program files\Razorcat\tessy_3.0\examples\IsValueInRange"!

Changing the interface of the test object

The target of this section is to show you the three different test readiness states "changed", "deleted" and "new".

Therefore we will first change a test object and add two new test objects called "delete" and "new". In a second step we will remove the "delete" object so it appears as deleted. The names are chosen to illustate the test readiness states.

→ Switch back to the Overview perspective.

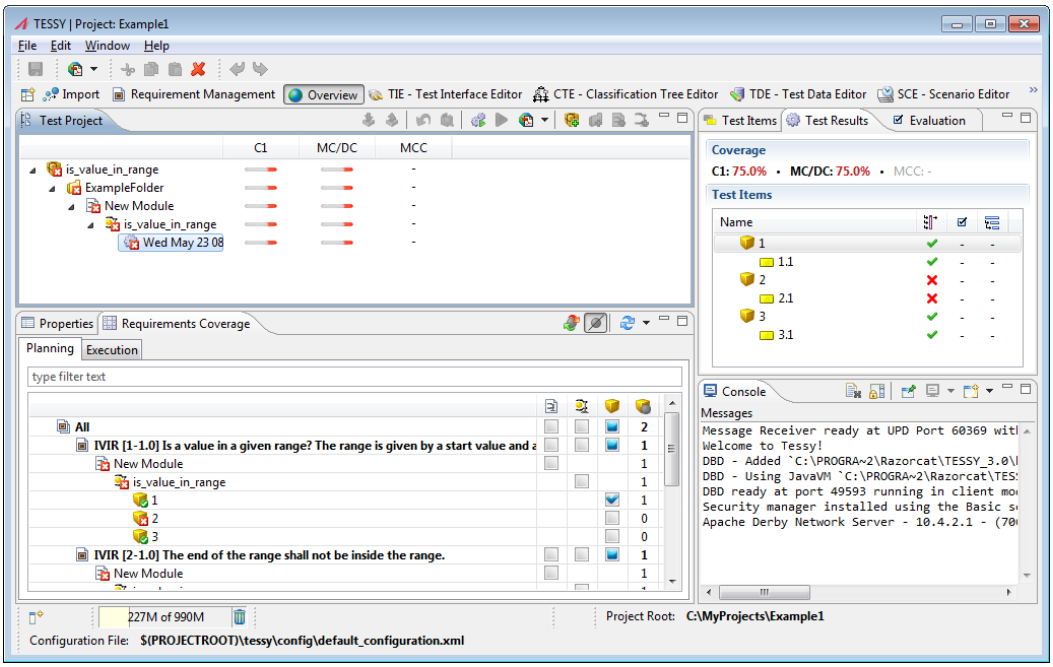


Figure 5.40: Overview perspective after test run (with requirements)

→ Select the module and "Edit Source" from the context menu (see figure 5.41).

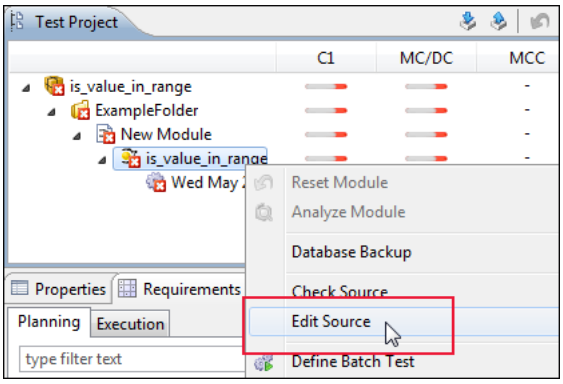


Figure 5.41: Use the context menu to edit a source

The C-source is opened.

→ Select the line "***result is_value_in_range (struct range r1; value v1)***" (see figure 5.42).

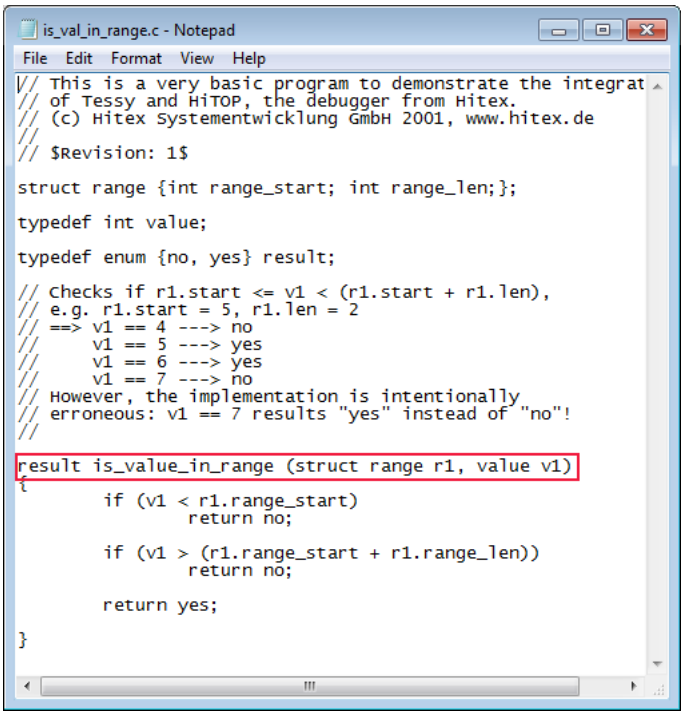


Figure 5.42: Editing the C-source file is_val_in_range.c

Changing the C-source

→ Edit the line as shown in figure 5.43

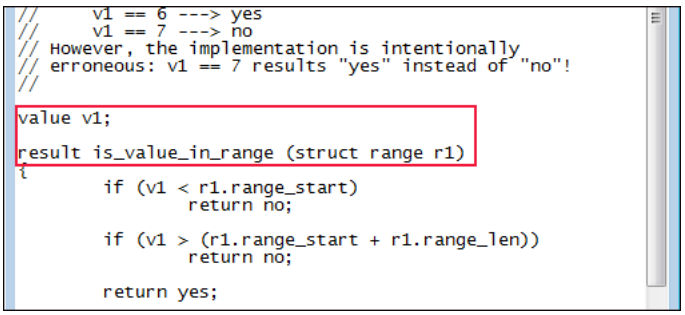


Figure 5.43: Changed C-source file of is_value_in_range

→ Now add a "delete" object and a "new" object as shown in figure 5.44

```
//
value v1;
result is_value_in_range (struct range r1)
{
        if (v1 < r1.range_start)
                return no;

        if (v1 > (r1.range_start + r1.range_len))
                return no;

        return yes;

}
void deleted(int p)
{}

void new()
{}
```

Figure 5.44: Adding a "delete" and "new" object

→ Save the changes with "File" > "Save" and close the file.

→ Click on to analyze the module.

In the Test Project view you can see now three test objects with different test readiness states (see figure 5.45):

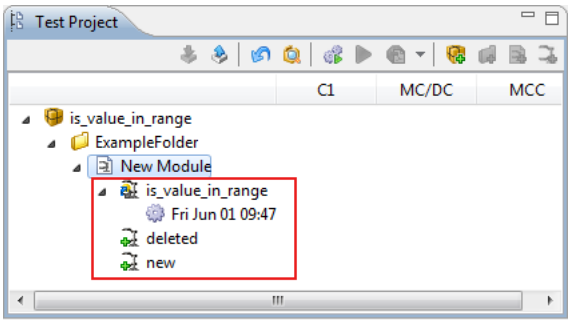


Figure 5.45: Changed and new test objects of is_value_in_range

The test object is_value_in_range has **changed**. You see the test object, but there is no operation possible. You have to start a reuse operation.

The test objects "deleted" and "new" are **newly** available since the last interface analysis. You have to add test cases, test steps and enter data for a test.

Before deleting the test object "deleted", we will have to add some test cases with test steps:

Deleted test objects that did not contain any test cases and test steps are not displayed anymore because they are considered as not important. If you want to display a deleted test object, you have to add at least one test case and one test step!

→ Switch to the Test Item view and add a test case and a test step.

→ Switch to the Overview perspective and to the Test Project view.

→ Select the module and "Edit Source" from the context menu.

→ Remove the test object "deleted" as shown in figure 5.46.

```
//
value v1;
result is_value_in_range (struct range r1)
{
        if (v1 < r1.range_start)
                return no;

        if (v1 > (r1.range_start + r1.range_len))
                return no;

        return yes;

}

void deleted(int p)
{}

void new()
{}
```

Figure 5.46: Remove the code for test object "deleted".

→ Save the changes with "File" > "Save" and close the file.

→ Click on to analyze the module.

In the Test Project view you can see now three test objects with three different test readiness states (see figure 5.47):

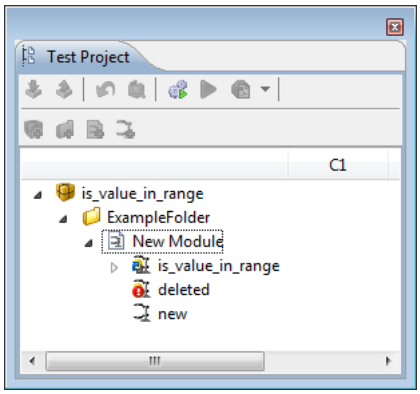


Figure 5.47: Changed and new test objects of is_value_in_range

- The test object is_value_in_range is still displayed as **changed** since there was no reuse operation yet.
- The test object "deleted" has been removed. You still see the object, but there is no operation possible.
- The test object "new" is not shown anymore as "newly available", because the last interface analysis already detected the object as new.

Important: Note that removed and changed test objects require a reuse operation before you can further operate on them!

Assigning the changed interface of the test object

Warning: If you do not assign the interface object, you will loose the test data entered for parameter v1 and the global variable v1 will have no values after the reuse operation!

→ Switch to IDA perspective.

→ Double click the test object "is_value_in_range" in the Test Project view to assign its interface.

Please notice (see figure 5.48):

- On the right side within the IDA perspective you see the Compare view with the test object is_value_in_range.
- Within the Compare view you can see the old interface of our test object is_value_in_range and the new one. The red exclamation mark within the new interface indicates the need to assign this interface object before starting the reuse.
- The title of the view shows the old name versus the newly assigned name. In our case the names are the same since only the interface did change.

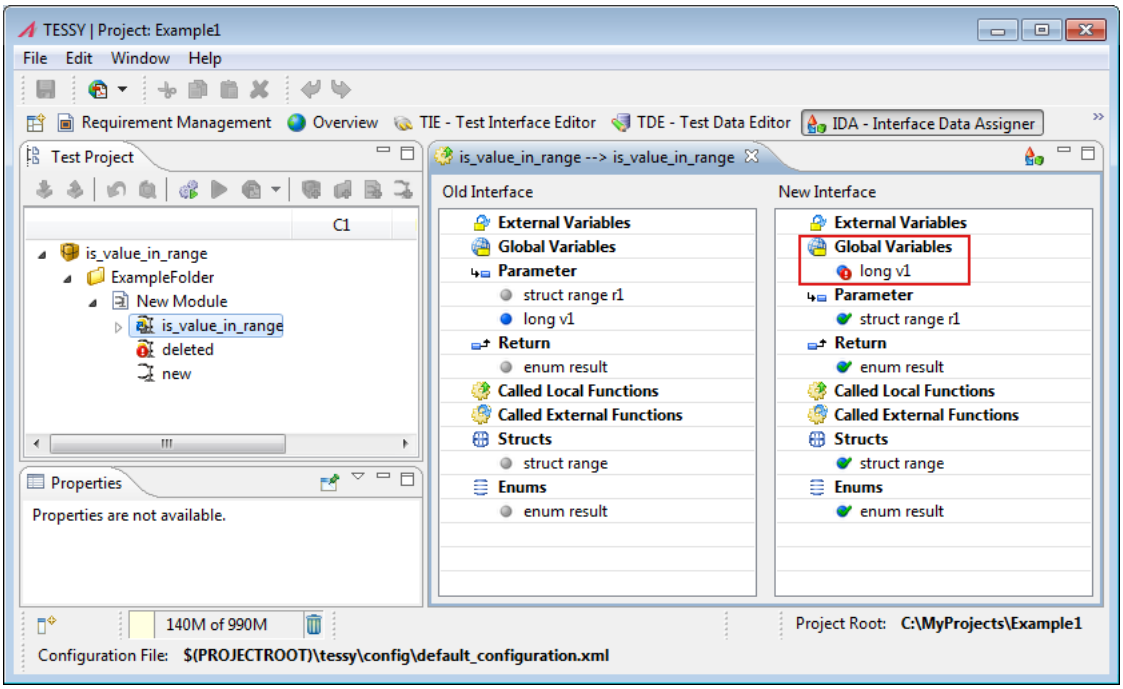


Figure 5.48: Changed, deleted and new test object of is_value_in_range

→ Assign the interface object "long v1" either by using the context menu or just drag and drop from the left side (see figure 5.49).

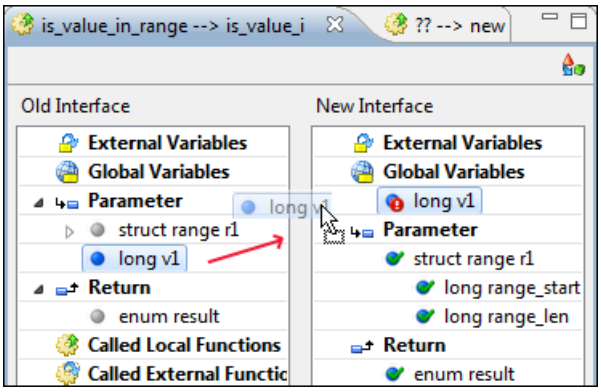


Figure 5.49: Use drag and drop in IDA

The red exclamation mark turns to a green check .

→ Commit the assignments by clicking on (Commit) in the menu bar of the Compare view.

The data of all test cases and test steps will be copied from the old interface to the current test object interface.

The test object changes to yellow to indicate that all test cases are ready to be executed again.

Please notice the following habits:

- Removed and changed test objects require a reuse operation before you can further operate on them.
- Those test objects that remained unchanged will automatically be reused, e.g. they will be ready to use without further activities required.
- Removed test objects will only be displayed as “removed”, if they did contain any test cases and test steps.

5.2 Quickstart 2: Exercise Classification Tree Editor (CTE)

To understand the handling and create a simple classification tree we consider some aspects from the Quickstart 1: Exercise is_value_in_range.

We will continue with the quickstart example "is_value_in_range". If you have not done the exercise, proceed with the Quickstart 1: Exercise is_value_in_range up to section Designing test cases.

After editing the test object interface and switching to the CTE, the root "is_value_in_range" of the classification tree appears on the draw pad.

Important: If you cannot see the root on the draw pad, move the scrollbar at the bottom or enlarge the window.

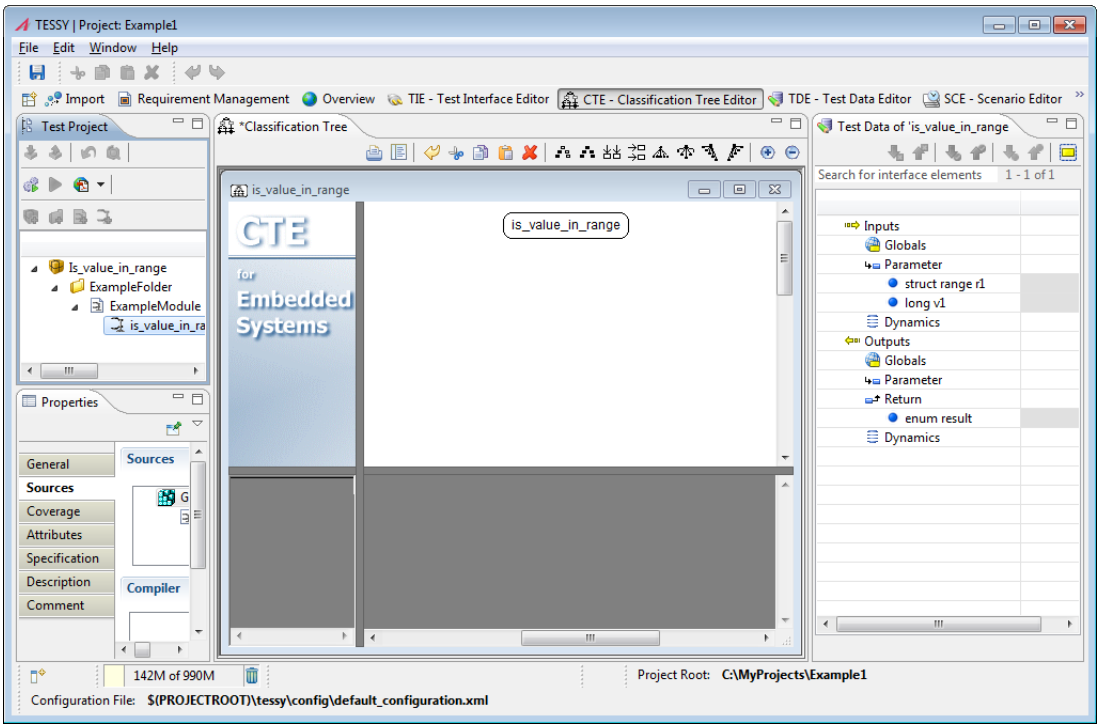


Figure 5.50: CTE with the root "is_value_in_range"

Example is_value_in_range

Our three test-relevant aspects are the start value of the range, the length and the position, which is a given value under test (v1). We should have test cases which result in "inside the range", and other test cases which result in "outside", to have a high coverage of possibilities.
So the three test-relevant aspects to be used for classifications are start value, length and position and they thus form the basis of a classification tree.

In the 'Advanced Knowledge' of this manual you find detailed information of using The Classification Tree Method (CTM).

5.2.1 Creating classifications

→ Select the root tree item ("is_value_in_range").

→ Create a new classification either using the context menu ("New" > "Classification") or press Ins (see figure 5.51).

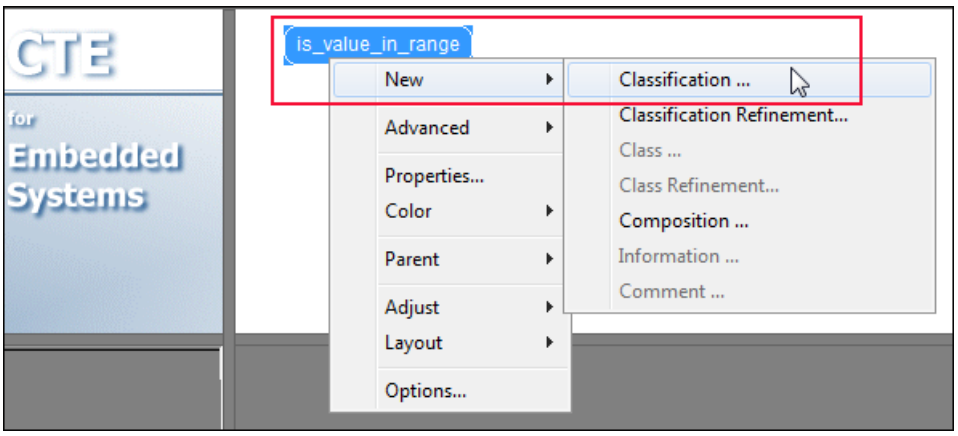


Figure 5.51: Creating a new classification with the context menu

→ Double click the new classification or press F2 to start the inline editor for the tree item.

→ In the window Element Properties rename the new classification to *range_start*.

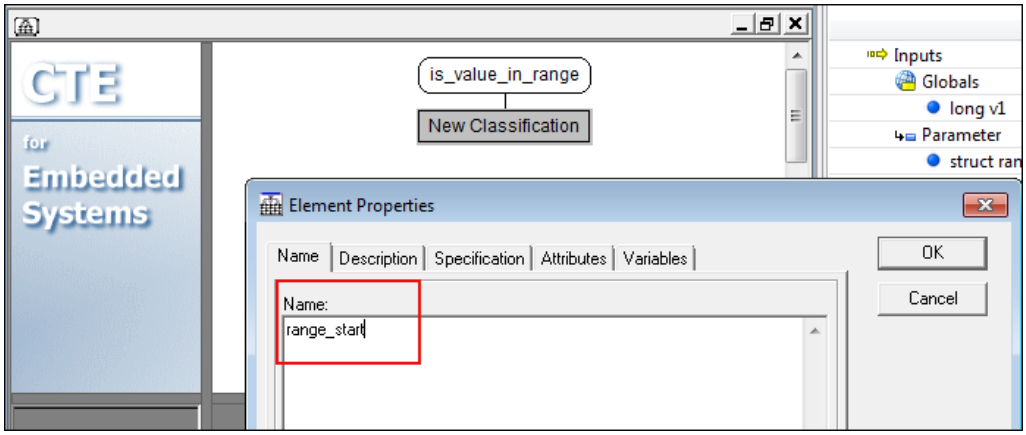


Figure 5.52: Renaming the new classification

→ Create two further classifications *range_length* and *position*.

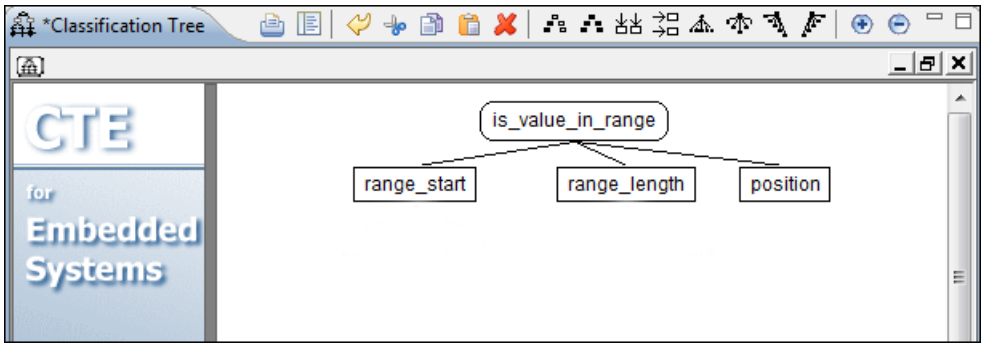


Figure 5.53: Classifications of is_value_in_range

Within the draw pad you can move the classifications with drag and drop: Select either a classification, a sub tree (click on) or select all (click on) and press the mouse button until the cursor turns into a cross with four arrows , then move the selection.

5.2.2 Creating classes

→ Now create the classes "positive", "zero" and "negative" for the classifications range_start und range_length: Right click the classification and select "New" > "Class" or select the classification and press Ins.

→ Create the classes "inside" and "outside" for the classification "position".

→ In the tool bar click on (Arrow Layout) to rearrange the tree for a better overview.

Try the several layouts for the tree, e.g. leftdown or horizontal may give you a better overview.

Now the classification tree should look like in figure 5.54.

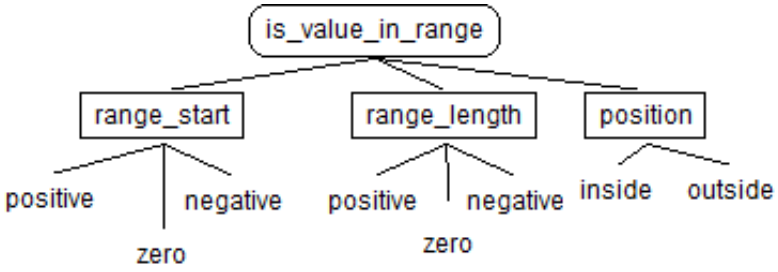


Figure 5.54: Simple classification tree for is_value_in_range

5.2.3 Entering test data

→ Select the class "positive" of the classification "range_start".

→ In the Test Data view on the right enter the value 3 as range start (see figure 5.55).

Important: To enter the data it might be necessary to enlarge the window or to change the width of the first column containing the interface elements.

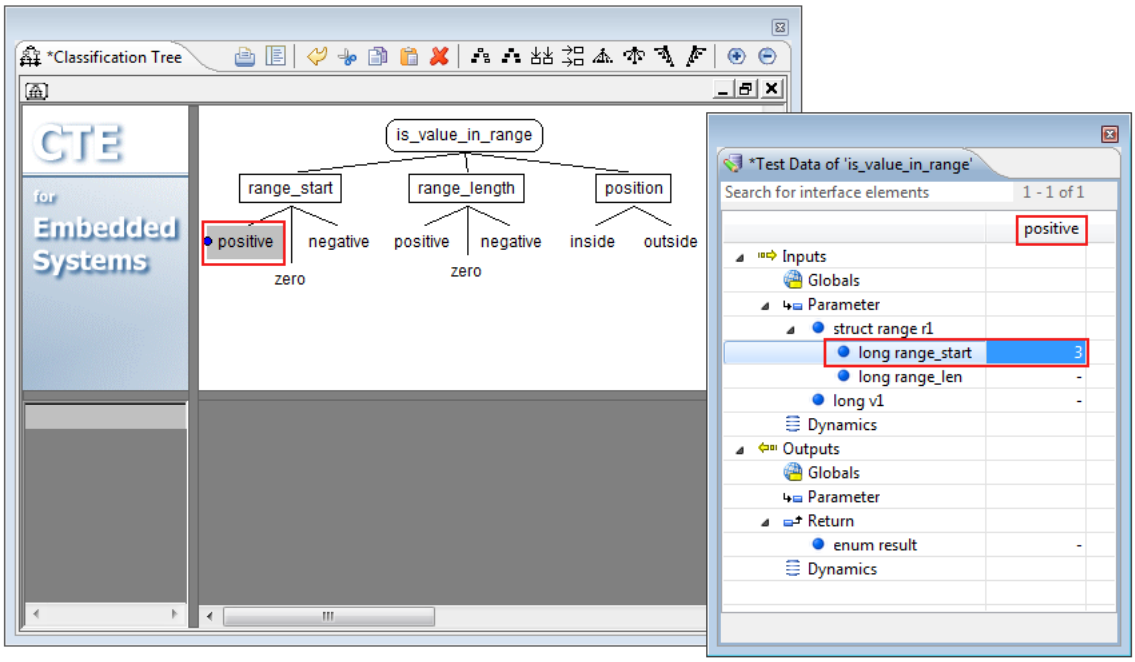


Figure 5.55: Entering data in CTE

Now enter more test data for the other classes of the classification "range_start":

→ Select the class "zero" and enter 0 as range_start.

→ Select the class "negative" and enter -3 as range_start.

When selecting a tree item within the classification tree you will see the test data entered for this tree item within the Test Data view.

Notice the following habits:

- The name of the selected item will be displayed as column header in the Test Data view (see red marked sections in figure 5.55, here: "positive").
- The class will be marked with a blue dot (when selected) to indicate that test data is assigned.
- All tree items with test data assigned will be marked by a yellow dot when not selected (e.g. select the root "is_value_in_range").

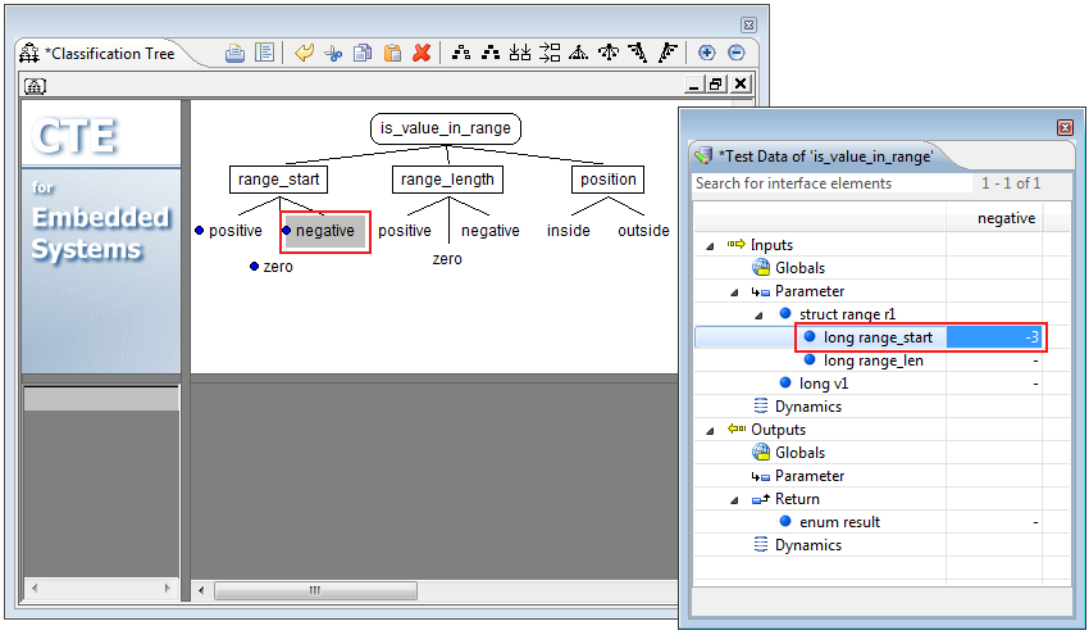


Figure 5.56: Entering more data in CTE

Entering values for "range_length"

Now enter test data for the classes of the classification "range_length":

→ Select the class "positive" of the classification "range_length" and enter 2 as range_length.

→ Select the class "zero" and enter 0 as range_length.

→ Select the class "negative" and enter -2 as range_length.

Notice the following habits:

- All tree items with assigned test data are marked with a yellow dot, when not selected.
- When selecting a tree item, you will see the test data entered for this item within the Test Data view.
- When selecting any interface element within the Test Data view, all classification tree elements that contain test data for this interface element will be marked with a blue dot.

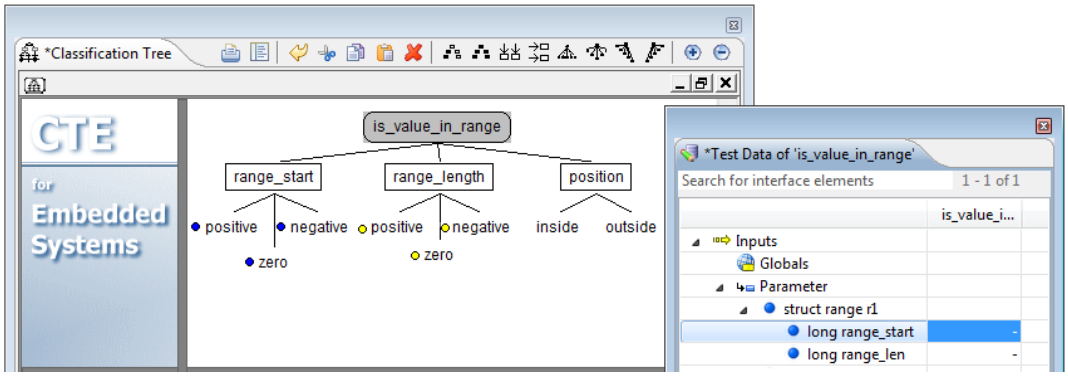


Figure 5.57: Blue dots indicate that "range_start" elements contain data

5.2.4 Creating test cases

Now create 9 test cases:

→ Select the pane on the lower left (test item list).

→ Create the test cases either using the context menu ("New" > "Classification", see figure 5.58) or press Ins.

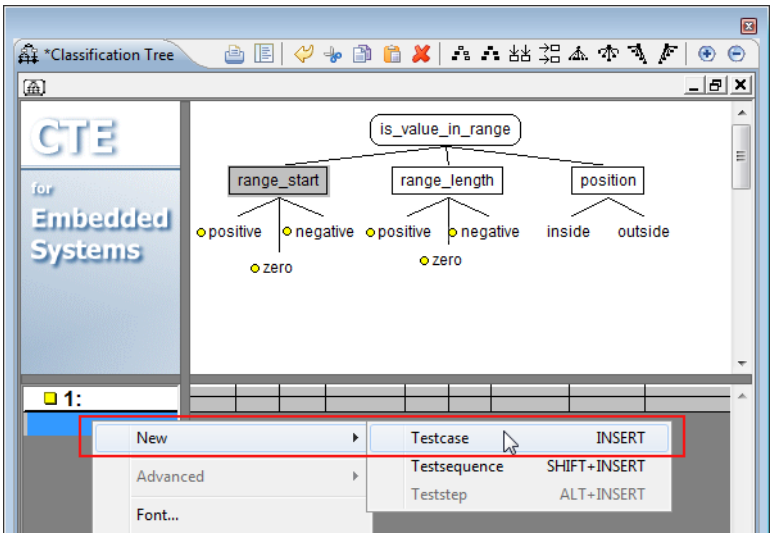


Figure 5.58: Creating test cases in the test item list

Test cases are defined by setting markings in the combination table: If you move the mouse pointer over the combination table, connecting lines to the classes of the classification tree are drawn. If the mouse pointer is placed over a point of intersection it is

changed to a circle.

→ Select the first test case in the test item list.

→ Move the mouse over the line of the first test case.

→ Click on the circles that connect the first test case with the two positive classes.

The test data for the test case 1 is displayed within the Test Data view (see figure 5.59). The test data is read-only because it is defined by the marks set within the combination table.

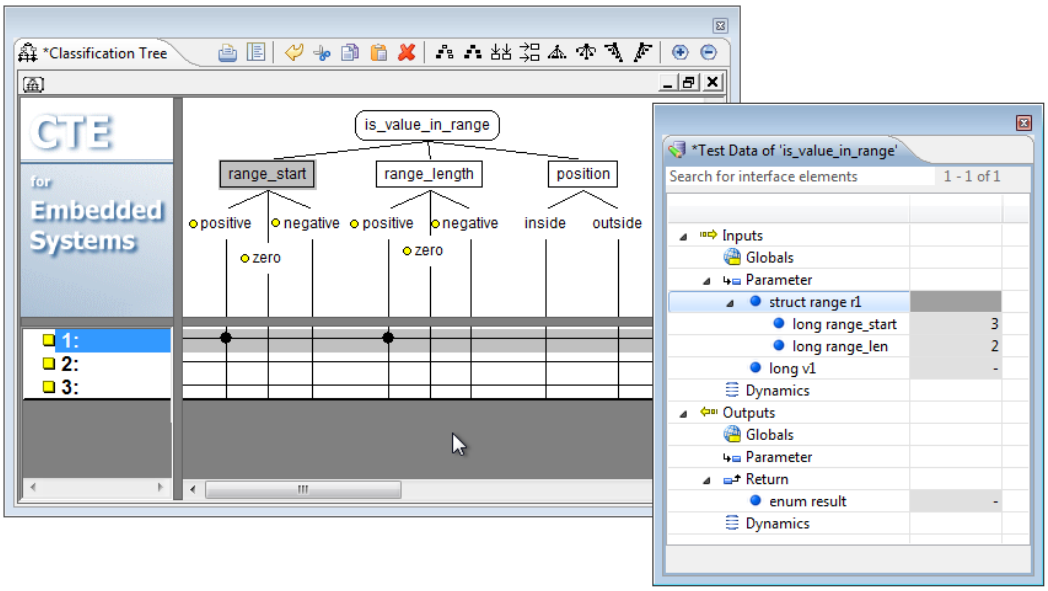


Figure 5.59: Defining test cases in the combination table of CTE

→ Create more marks within the combination table for the other test cases to cover completely all classes of the classifications "range_start"' and "range_length" with your test cases.

For example the CTE could look like in figure 5.60.

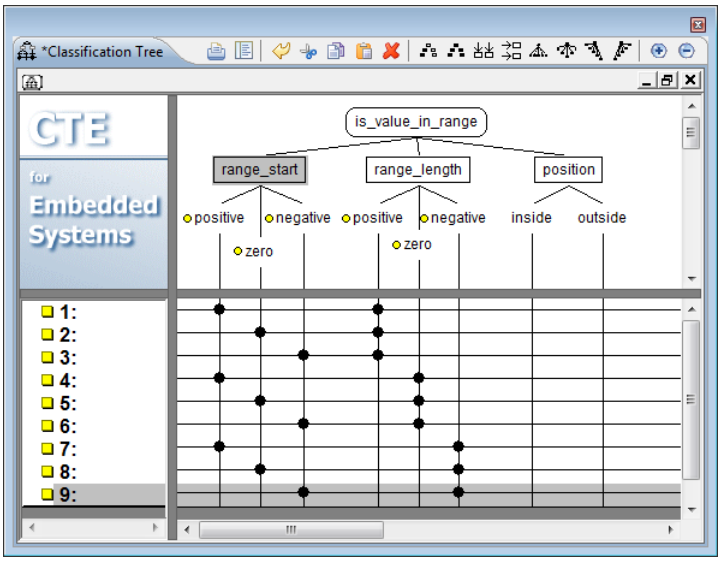


Figure 5.60: Completed table with all test cases for example "is_value_in_range"

Certainly there are various possibilities of combining the classes within the test cases.

As you can see the classification "position" is still not marked for any test item. This would be rated as an error because each class of the classification tree should be marked at least within one test item.

We also did choose all possible combinations for the first two classifications. In a real example you would need to select only the most interesting combinations in order to get a reasonable number of test cases.

→ Select the test cases one after another and review the test data resulting from your mark settings being displayed within the Test Data view (see figure 5.61).

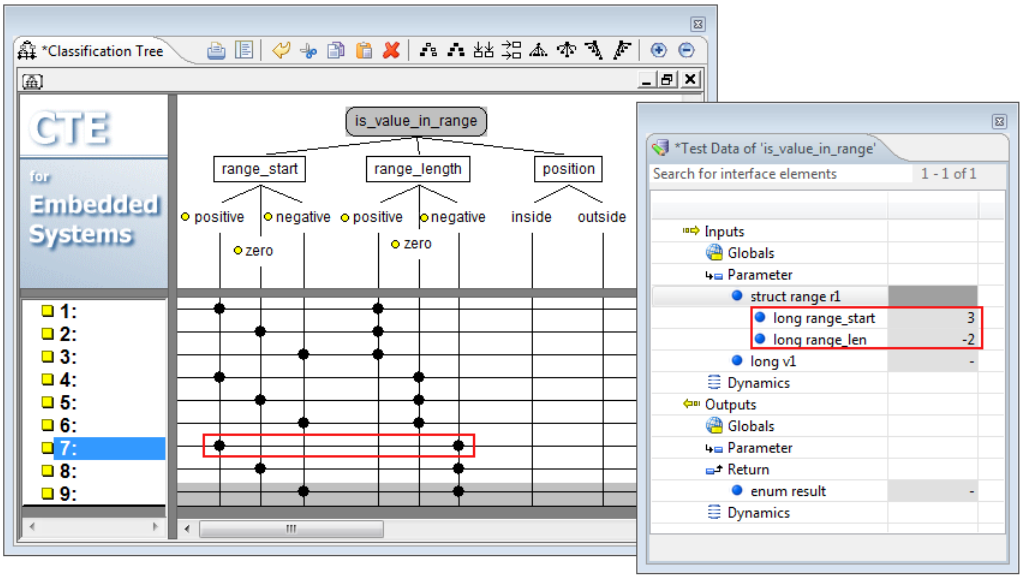


Figure 5.61: Test data is displayed when selecting a test case in the combination table

→ Click on to save the classification tree.

→ Switch to the TDE perspective.

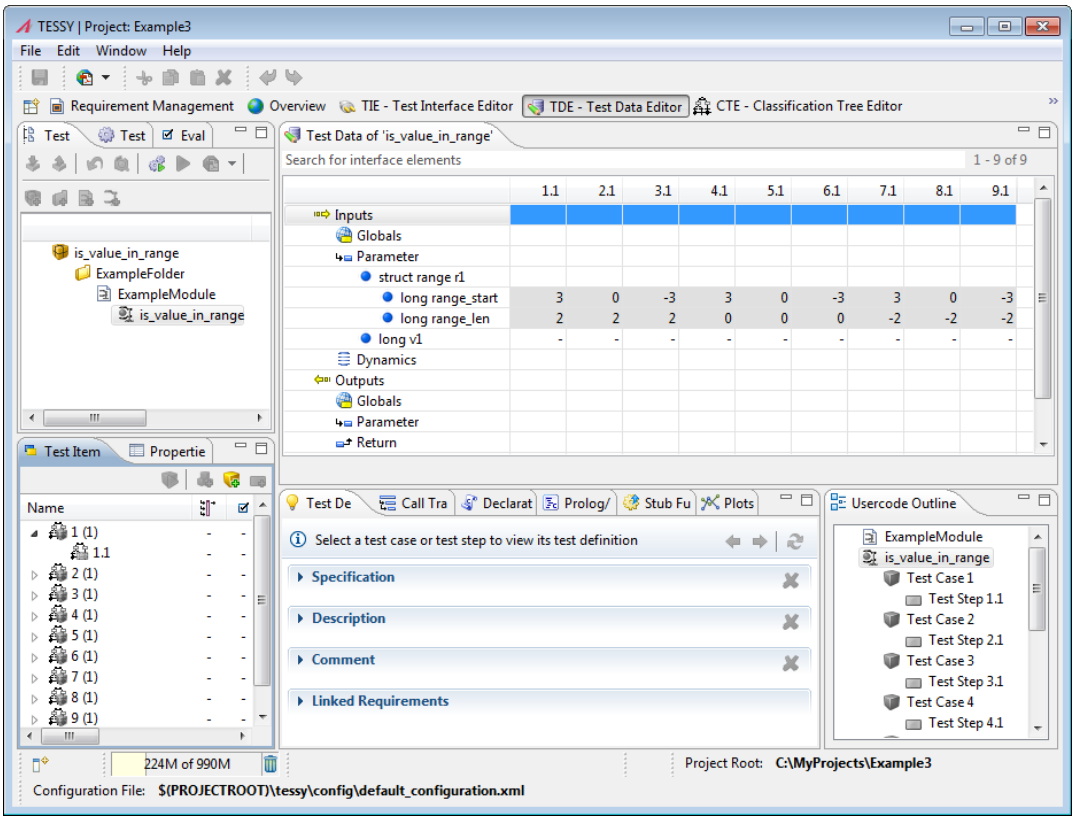


Figure 5.62: Test data displayed within TDE

You will see the test cases updated with the test data values entered within the CTE perspective.

Please notice the following habits:

- Test items with values stemming from the CTE perspective are marked with new status indicators: (test case) and (test steps).
- Values stemming from the CTE are read-only. If you want to change them, switch back to the CTE perspective and do your changes there.

6 Working with TESSY

This chapter provides detailed information of the test activities possible with TESSY. The headlines of the sections follow the actions taken during a test and refer to the corresponding perspectives and views, e.g. "CTE: Designing the test cases".
The subsections describe the views of each perspective, displaying used icons and status indicators and giving quickly answers to your questions of "What can I do within this view?" and "How do I do . . . ?". So if you need help at some point, ask "Where am I?". You should find the answer easily within this chapter. If you have questions about the workflow, consult chapter 5 Practical exercises.

Some views are displayed within various perspectives. Because views are context sensitive, not every operation is possible within every perspective. In this case the manual will refer to the respective section and perspective, where all operations of the view are possible.

6.1 Overview perspective: Organizing the test . . . 117
6.1.1 Structure of the Overview perspective . . . 117
6.1.2 Test Project view . . . 118
6.1.3 Properties view . . . 134
6.1.4 Requirements Coverage view . . . 143
6.1.5 Test Items view . . . 143
6.1.6 Test Results view . . . 147
6.1.7 Evaluation Macros view . . . 147
6.1.8 Console view . . . 148
6.1.9 Suspicious Elements view . . . 149
6.2 Import perspective: Importing requirements . . . 150
6.2.1 Importing requirements . . . 150
6.2.2 Formats . . . 152
6.3 Requirement management . . . 153
6.3.1 Structure of the Requirement Management perspective . . . 154
6.3.2 RQMT Explorer view . . . 155
6.3.3 Requirements List view . . . 160

6.3.4 Test Means view . . . 162
6.3.5 VxV Matrix view . . . 163
6.3.6 Link Matrix view . . . 164
6.3.7 Suspicious Elements view . . . 167
6.3.8 Attached Files view . . . 170
6.3.9 Attributes view . . . 171
6.3.10 History view . . . 173
6.3.11 Related Elements view . . . 175
6.3.12 Requirements Coverage view . . . 175
6.4 TEE: Configurating the test environment . . . 179
6.4.1 Starting the TEE . . . 180
6.4.2 Structure and handling of the TEE . . . 180
6.4.3 Status indicators . . . 182
6.4.4 Configuration files . . . 184
6.4.5 Using default configurations . . . 185
6.4.6 Enable configurations . . . 186
6.4.7 Customizing enabled configurations . . . 188
6.5 TIE: Preparing the test interface . . . 191
6.5.1 Structure of the TIE perspective . . . 192
6.5.2 Test Project view . . . 192
6.5.3 Properties view . . . 192
6.5.4 Interface view . . . 193
6.5.5 Plots view . . . 201
6.6 CTE: Designing the test cases . . . 202
6.6.1 The basic idea . . . 202
6.6.2 Starting CTE . . . 202
6.6.3 Structure of the CTE perspective . . . 203
6.6.4 Test Project view . . . 203
6.6.5 Properties view . . . 203
6.6.6 Classification Tree view . . . 204
6.6.7 Test Data view . . . 209
6.7 TDE: Entering test data . . . 210
6.7.1 Structure of the TDE perspective . . . 210
6.7.2 Test Project view . . . 211
6.7.3 Test Results view . . . 211
6.7.4 Evaluation Macros view . . . 212
6.7.5 Test Items view . . . 212
6.7.6 Properties view . . . 212
6.7.7 Test Data view . . . 213

6.7.8 Test Definition view . . . 224
6.7.9 Prolog/Epilog view . . . 225
6.7.10 Usercode Outline view . . . 230
6.7.11 Plots view . . . 231
6.8 CV: Analyzing the Coverage . . . 236
6.8.1 Structure of the CV perspective . . . 238
6.8.2 Test Project view . . . 238
6.8.3 Called Functions view / Code view . . . 238
6.8.4 Flow Chart view . . . 239
6.8.5 Branch (C1) Coverage view . . . 243
6.8.6 MC/DC Coverage view . . . 244
6.8.7 MCC Coverage view . . . 245
6.8.8 Coverage Report views . . . 245
6.9 IDA: Assigning interface data . . . 246
6.9.1 Structure of the IDA perspective . . . 247
6.9.2 Test Project view . . . 247
6.9.3 Properties view . . . 248
6.9.4 Compare view . . . 248
6.10 SCE: Component testing . . . 251
6.10.1 Creating component tests . . . 252
6.10.2 Preparing the test interface . . . 253
6.10.3 Designing the test cases . . . 254
6.11 Backup, restore, version control . . . 255
6.11.1 Backup . . . 255
6.11.2 Restore . . . 256
6.11.3 Version control . . . 257
6.12 Command line interface . . . 258
6.12.1 Invoking tessycmd.exe . . . 258
6.12.2 Usage of tessycmd.exe . . . 259
6.12.3 Commands . . . 259
6.12.4 Example: DOS script . . . 260

6.1 Overview perspective: Organizing the test

Important: If you have not created a project yet, do so as described in the chapter "Basic handling" in section 3.1.1 Creating a project database!

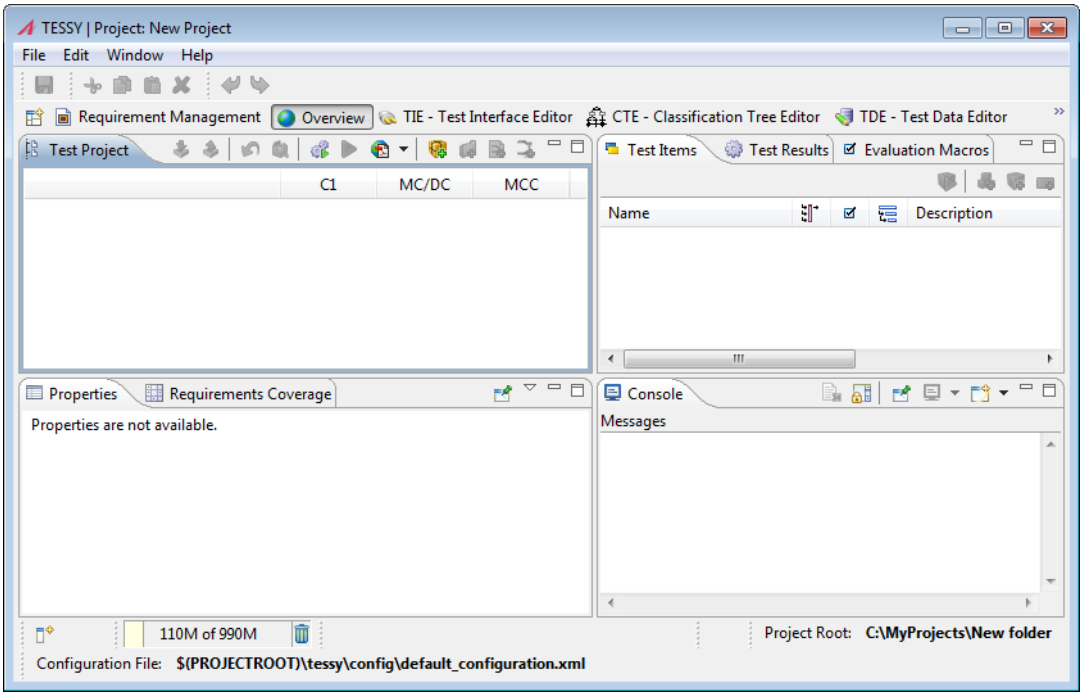


Figure 6.1: Overview perspective

6.1.1 Structure of the Overview perspective

pane	**location** (default)	**function**
Test Project view	upper left	To organize the project: Create test collections, modules and test objects; execute the test, create reports and have a fast overview on your project.
Properties view	lower left	To edit all properties, e.g. adding sources or including paths to your modules.

continue next page

pane	**location** (default)	**function**
Requirement Coverage view	lower left	To select and link the requirements that you managed within the Requirement management perspective.
Test Items view	upper right	To create test cases and test steps manually.
Test Results view	upper right	To view the test results.
Evaluation Macros view	upper right	To view evaluation macro results, if the usercode of the test object contains any.
Console view	lower right	To display messages of sub processes invoked during test execution, e.g. compiler calls.
Suspicious Elements view	lower right	To review changes of requirements, modules or test objects that require updating the linked test cases.

Table 6.1: Structure of the Overview perspective

6.1.2 Test Project view

Test Project view

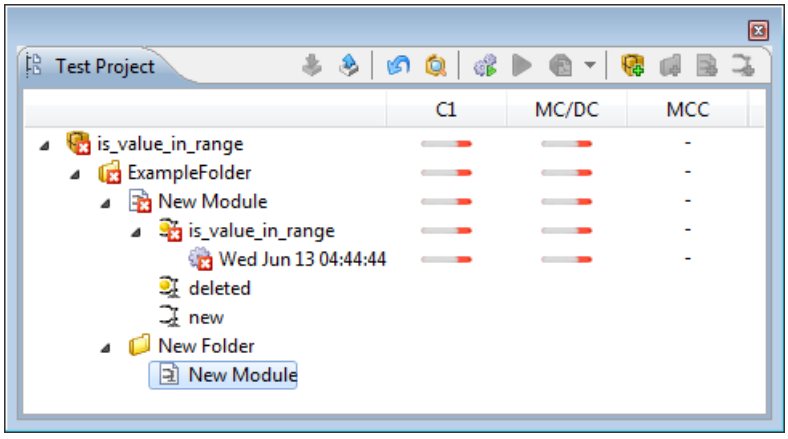


Figure 6.2: Test Project view within the Overview perspective

Icons of the view tool bar

icon	action / shortcut / comment
	Imports files (type depends on the selection).
	Exports files.
	Resets a module (deletes all contents of the module!).
	Analyzes the C-source file(s) of the module.
	Defines a batch test.
	Executes the test (Ctrl + E).
	Generates various test reports. The test details report for a test object will be generated as default (Ctrl + R).
	Inserts a new test collection.
	Inserts a new folder (Shift + Ins), optional for organizing your test project.
	Inserts a new module (Ins). Modules will contain the test objects available within the C-source files to be tested, i.e. C functions.

Table 6.2: Tool bar icons of the Test Project view

Status indicators

indicator	status / meaning
	Test collection containing folders (optional) and modules.
	Folder containing modules (optional).
	Module containing the test objects.
	The test objects are analyzed but have no test case.
	The test objects have test cases but no data.
	At least one test case is ready to be executed.

continue next page

indicator	status / meaning
	The test object interface has changed. A reuse operation within IDA is required.
	The test object has been removed. You still see the object, but there is no operation possible. Only displayed when the test object contained any test cases before the removal.
	The test object is suspicious.
	The test execution has been aborted for this test object.
	The test result of a test run is failed. This may be either due to a mismatch of actual and expected results or if the coverage did not achieve the minimum coverage.
	Test results and coverage of the test run are both OK.
	The coverage did not achieve the required minimum coverage. The red part of the bar indicates the missing percentage of coverage, e.g. more red means less achieved coverage.
	The coverage achieved the minimum coverage, but the minimum coverage was less than 100.

Table 6.3: Status indicators of the Test Project view

Creating tests

Creating the test

You need at least one test collection to organize your test, and within at least one module and one test object. Folders and further test collections are optional and just have the purpose to organize your test project.

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

→ Click on the icon (New Test Collection) on the tool bar of the view.

→ Enter a name and click "OK".

→ Click on (New Folder), enter a name and click "OK".

→ Click on (New Module), enter a name and click "OK".

Analyzing modules

Important: To analyze a module it is necessary to select an execution environment, add at least one C-source file and required include paths for header files and add defines necessary for analyzation/compilation of the source file(s), see section 6.1.3 Properties view. If you need to learn more about this workflow, have a look at the Practical exercises.

Analyzing the C-source file

To analyze a module (the C-source file),

→ in the tool bar click on (Analyze Module) to start the module analysis.

TESSY now analyzes the C-source file, this will take a few seconds.
After successful processing,

→ click on the white arrow in front the module: ExampleModule.

TESSY will as well analyze the C-source file by just clicking on the white arrow next to the module after adding the C-source-file.

Now all functions which were defined in the C-source file are displayed as children of the module above within the Test Project view (see figure 6.3).

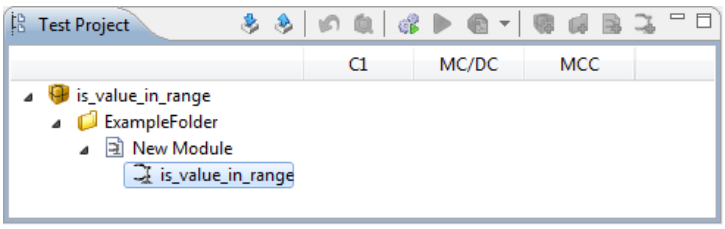


Figure 6.3: The function of the C-source is displayed as child of the module.

Executing tests

After entering test data for a particular test object you are ready to execute the test. During this process, TESSY will perform the following steps:

- Generate the test driver based on the interface information and user code provided.
- Link the test driver to the test object to create an executable file.
- Run the test.

Generating the test driver

The test driver is necessary to call the function under test and TESSY generates the driver automatically. The test driver and the function under test together form a complete (embedded) test application, including the startup code for it. In case the function under test uses external variables that are not defined, the test driver generated by TESSY can define those variables in the test driver. Similarly, if the function under test itself calls other functions (subroutines), TESSY can provide replacement functions (stubs) for the missing subroutines with the test driver. TESSY features two types of stub functions:

- Stub functions for which you may provide the C source code for the bodies of the stub functions.
- Stub functions for which TESSY is able to check if the expected value for a parameter is passed into the stub function and for which TESSY is able to provide any value specified by the user as return value of the stub function.

The test driver, the function under test, stubs and everything else is automatically compiled and linked by TESSY using an appropriate compiler for the particular embedded microcontroller architecture being used.

Once the test driver has been built and compiled, it can be run as often as required. You can select a subset of your test cases and run the test again by just selecting the run option. Changes to test data and expected results might require building a new test driver. TESSY will check this automatically and generate a new driver.

Executing the test

To execute a test:

→ Click on the arrow next to the icon Start Test Execution in the tool bar of the Test Project view.

A progress dialog will be opened (see figure 6.4).

→ Choose the desired options and click "Execute".

A progress dialog will be shown while TESSY generates, compiles and links the test driver and runs the test. This will take a few seconds.

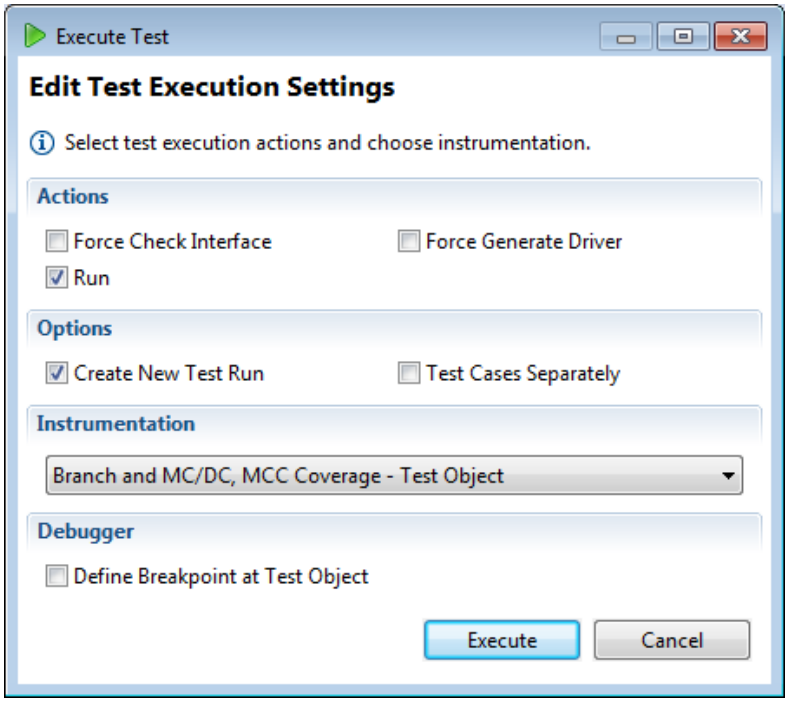


Figure 6.4: Test Execution Settings

Instrumentation settings

TESSY supports both C1 and C2 coverage measurements. Since there are different meanings associated with these coverage measurement names, we will shortly explain our definition of these measurements:

We distinguish between C1 (branch coverage measurement), MCC (multiple condition coverage) and MC/DC (modified condition decision coverage). The last two measurements are referred to as C2 coverage measurement. All three measurements may be used in conjunction when running tests.
The MC/DC coverage includes the calculation of the required atomic value combinations for each condition within the test object. This helps developers in finding the required combinations for full MC/DC coverage. MC/DC in TESSY supports branch points such as in if or while statements.

Term	Explanation
None	No instrumentation will be applied.
C1 Test Object	Branch coverage instrumentation will be applied (only the test object code will be instrumented).
C1 Test Object and called functions	Branch coverage instrumentation will be applied for the test object itself and for all functions called by the test object.
C2 Test Object	MCC and/or MC/DC instrumentation will be applied for the test object (only the test object code will be instrumented).
C1 Test Object and called functions	MCC (multiple condition coverage) and MC/DC (modified condition decision coverage) instrumentation will be applied for the test object itself and for all functions called by the test object.
C1/C2	Both coverage measurements will be applied as described above.

Table 6.11: Coverage measurements

Viewing test results

Viewing test results

After a test run, the Test Project view gives an overview about the coverage, if selected:

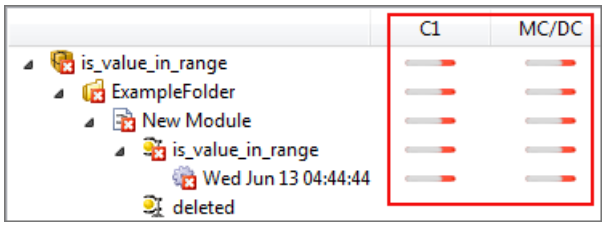
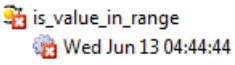


Figure 6.5: Coverage displayed within the Test Project view

- The actual results will be compared with the expected values according to the evaluation mode. The result will be either failed or passed.
- The last step of test execution is the generation of an XML result file. This file contains all test data and the actual results. It will be used for reporting.

The results of every coverage measurement can be reviewed in the CV (Coverage Viewer) as soon as the test was carried out. For details refer to section 6.8 CV: Analyzing the Coverage.

Please notice the following habits:

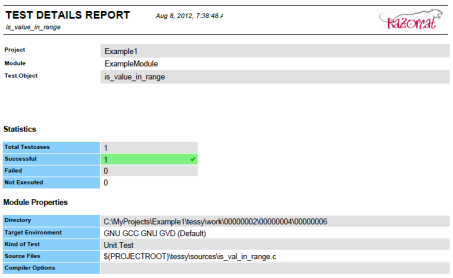
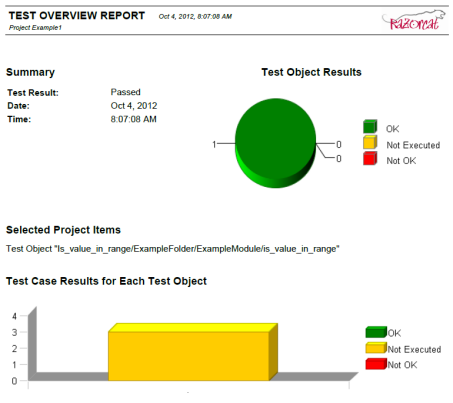
- A green tick will indicate that the actual value complies with the expected value for the given variable with respect to the evaluation mode.
- A red cross will indicate that the actual value does not comply with the expected value for the given variable with respect to the evaluation mode.
- If the interface has changed, the test object will indicate changes with test readiness states (see Status indicators).
- The time of the test run is stated within the Test Project view:
  

Important: The results of the coverage measurement are also part of the test result for a test object, e.g. if all outputs yield the expected result but the coverage was less than the minimum coverage, the test result will be failed.

Creating reports

Important: All reports are generated as PDF-files. You need the Adobe Reader to open the files. Download and install the Adobe Reader from http://get.adobe.com/reader/.

The following reports are available in TESSY:

Test Details Report  TEST DETAILS REPORT Project: Example1 Module: ExampleModule Test Object: is_value_in_range Statistics Total Testcases: 1 Successful: 1 Failed: 0 Not Executed: 0 Module Properties Directory: C:\MyProjects\Example1\tessy\work\00000002\00000004\00000006 Target Environment: GNU GCC GNU GVD (Default) Kind of Test: Unit Test Source Files: ${PROJECTROOT}/tessy/sources/is_val_in_range.c Compiler Options:	Contains information about the test cases, their properties and values, linked requirements and if test cases failed or were successful. The Test Details report is based on the XML result file which is created after every test run and stored within the latest test run. A test report can be generated as soon as TESSY has finished running a test.
Test Overview Report  TEST OVERVIEW REPORT Summary Test Result: Passed Date: Oct 4, 2012 Time: 8:07:08 AM Test Object Results OK Not Executed Not OK Selected Project Items Test Object "Is_value_in_range/ExampleFolder/ExampleModule/is_value_in_range" Test Case Results for Each Test Object	Contains information about the test objects and the test case results in grafical form (pie chart). The Test Overview Report summarizes the results of a selected set of test objects. The report can be generated based on already executed test objects or right after the execution of the selected test objects.

continue next page

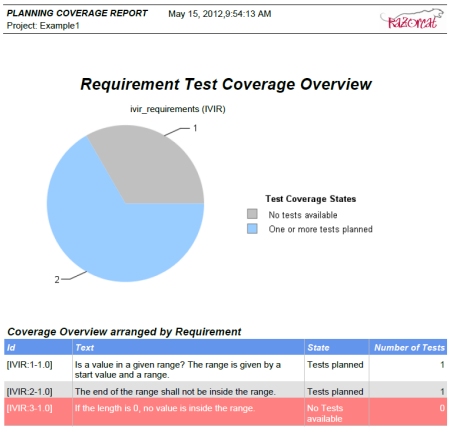
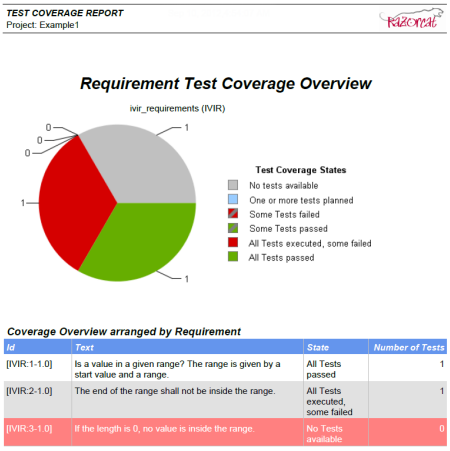
Planning Coverage Report  PLANNING COVERAGE REPORT May 15, 2012,9:54:13 AM Project: Example1 Requirement Test Coverage Overview ivir_requirements (IVIR) Test Coverage States No tests available One or more tests planned Coverage Overview arranged by Requirement Id / Text / State / Number of Tests [IVIR:1-1.0] Is a value in a given range? The range is given by a start value and a range. Tests planned 1 [IVIR:2-1.0] The end of the range shall not be inside the range. Tests planned 1 [IVIR:3-1.0] If the length is 0, no value is inside the range. No Tests available 0	Contains information about the requirements linked to test cases and the planned validation. Planning coverage summarizes the achieved coverage of requirements by test cases. Each link of a test case to a requirement will be counted. This report provides information about the planning progress of the test project, since it will show all requirements that have not yet been planned to be tested by at least one test case.
Execution Coverage Report  TEST COVERAGE REPORT Project: Example1 Requirement Test Coverage Overview ivir_requirements (IVIR) Test Coverage States No tests available One or more tests planned Some Tests failed Some Tests passed All Tests executed, some failed All Tests passed Coverage Overview arranged by Requirement Id / Text / State / Number of Tests [IVIR:1-1.0] Is a value in a given range? The range is given by a start value and a range. All Tests passed 1 [IVIR:2-1.0] The end of the range shall not be inside the range. All Tests executed, some failed 1 [IVIR:3-1.0] If the length is 0, no value is inside the range. No Tests available 0	Contains information about the validation of requirements after test run. Execution coverage summarizes the achieved coverage of requirements based on executed tests. The results of test runs (i.e. the result of each test case) will be propagated to the list of requirements by creating requirement evaluations for all linked requirements, if any link was set. This report provides an overview about the result status of the test project since it will show all requirements for which the linked test cases are failed for any reason.

Table 6.15: Reports available within TESSY

All reports are created as PDF documents based on XML data files. These XML data files can also be used for generating reports or further processing if desired.

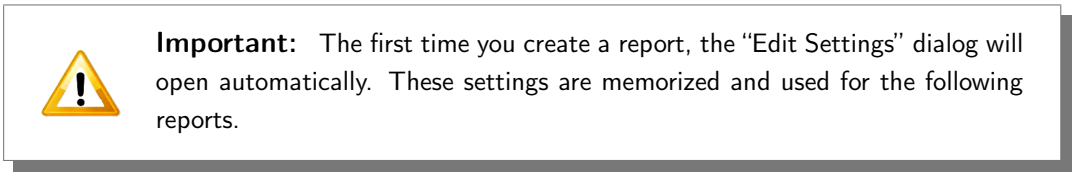
To create a report within TESSY: *Creating reports*

→ Click in the Test Project view (i.e. within the Overview perspective) on the arrow next to the Generate Report icon .

→ Select the report you wish to create (see figure 6.6).

TESSY creates the report within the new folder. This will take a few seconds.

When finished, TESSY will open the file automatically.



Important: The first time you create a report, the "Edit Settings" dialog will open automatically. These settings are memorized and used for the following reports.

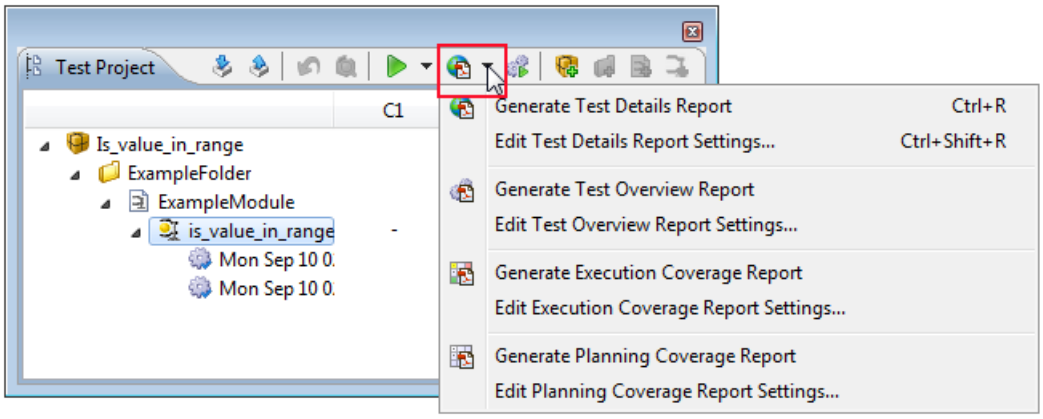


Figure 6.6: Creating a report

Change settings of a report

To change settings,

→ click on the arrow next to the Generate Report icon .

→ Select "Edit [report] Settings...".

the settings dialog for the selected report will be opened.

Optional settings of the different reports and their default values:

report	**settings**	**default**
Test Details Report	folder selection show usercode show test step details hide results (show test data only)	`C:\tessy\report` yes yes no
Test Overview Report	folder selection show test object result pie chart show test object list show test case results bar chart show coverage charts	`C:\tessy\report` yes yes yes yes
Planning Coverage Report	folder selection show overview pie chart show requirements list show test item list selection of requirement document selection of test means	`C:\tessy\report` yes yes yes selected, if only one available no
Execution Coverage Report	folder selection show overview pie chart show requirements list show test item list selection of requirement document selection of test means	`C:\tessy\report` yes yes yes selected, if only one available no

Table 6.18: Available settings of the reports

Changing the logo

To change the default Razorcat logo within the reports to your own company logo:

→ In the menu bar select "Window" > "Preferences".

→ Switch to the section General Settings.

→ "Test Report" > "LogoImage" (see figure 6.7)

→ Enter the path to your logo image file (PNG, JPG or GIF format).

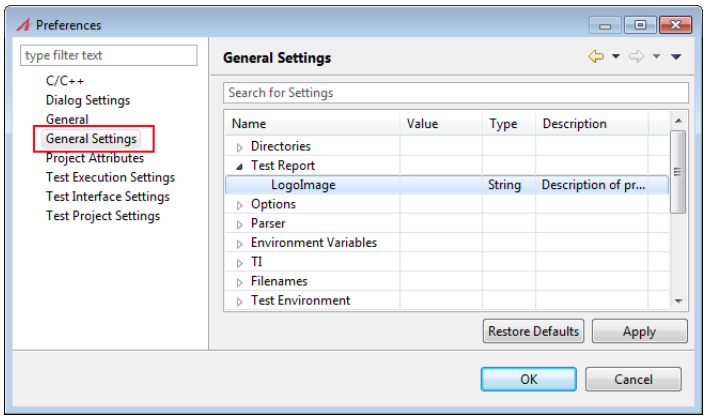


Figure 6.7: Changing the logo image for reports

Batch test operations

Batch Test

Tessy provides a batch test feature with various operations.

Open the batch operation settings:

→ In the Test Project view right click a project, a module or a test object.

→ Select "Define Batch Operation..." from the context menu (see figure 6.8).

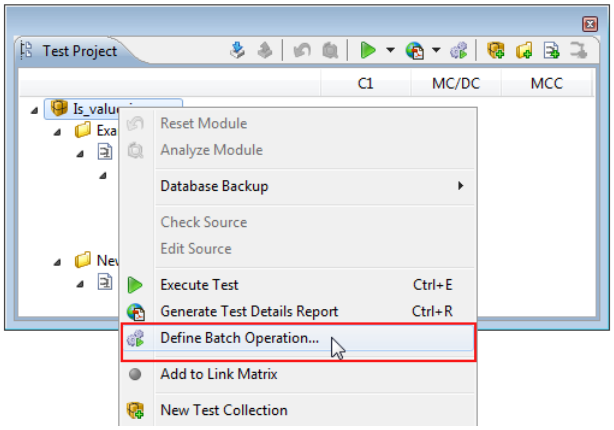


Figure 6.8: Defining the batch operation

The window "Define Batch Operation" will be opened.

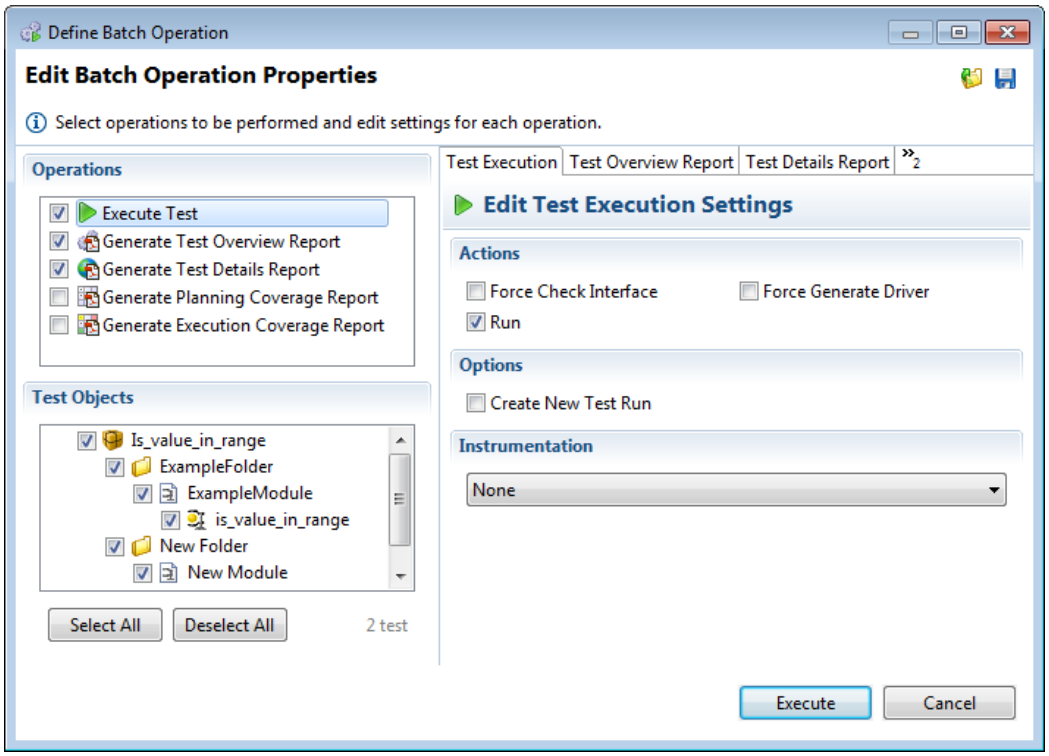


Figure 6.9: Defining the batch operation

To select the test objects for the batch test:

→ Under "Test Objects" choose the project or modules or test objects for the batch test. Click "Select All" to select all at once (see figure 6.10).

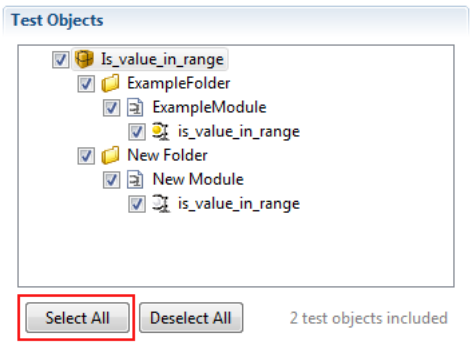


Figure 6.10: Defining the batch operation

The right side of the window ist context sensitive:

→ Switch to the setting by either marking the operation on the left side or use the tabs on the upper right side (see figure 6.11). The optional settings for this operation will then be shown on the right side of the window.

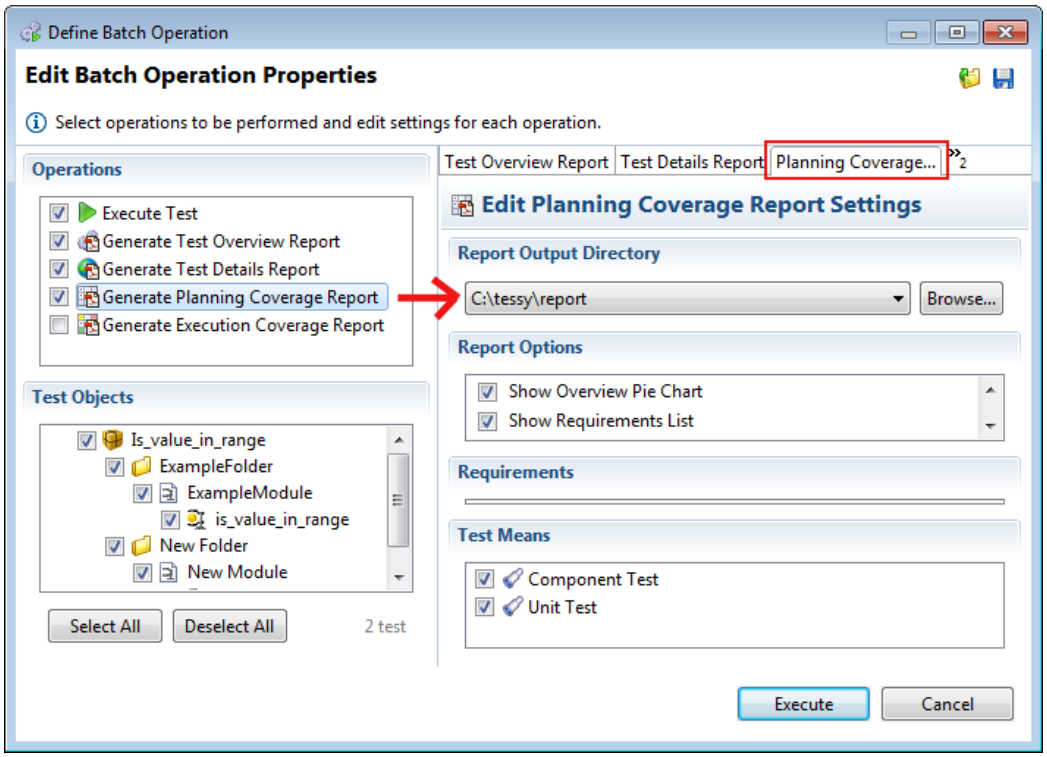


Figure 6.11: Editing the settings of each batch operation

Importing and Exporting

The Test Project view provides the import and export of test data and module backup files (*.TMB):

→ Select a folder, module or test object.

→ In the toolbar click on to import or to export the data.

The kind of import/export depends on the selection:

selection	type of file
folder	import: *.TMB files
module	export: TMB file
test object	import and export: test data

When importing data there are various options (see figure 6.12):

- "Update passing directions": If you tick the box, the passing directions of all interface variables will be set according to the passing directions specified within the import file. All other interface variables will be set to IRRELEVANT. The test object will be ready to execute when using this option because all variable with passing directions IN, OUT or INOUT will be filled with values.
- "Overwrite/append test cases": Either delete existing test cases before importing or append any imported test cases at the end of the test case list.

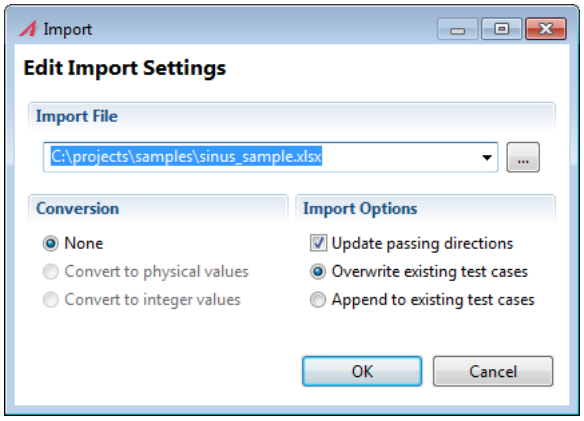


Figure 6.12: Import settings of data import

When exporting data there the following options (see figure 6.13):

- The Conversion of the export settings is only applicable, if ASAP conversion is enabled! Please refer to table "General tab og Properties view" in section General.

- Input/Expected versus Input/Actual: Either export the expected or the actual result values. The latter is only available if the test has been executed and actual results are present.

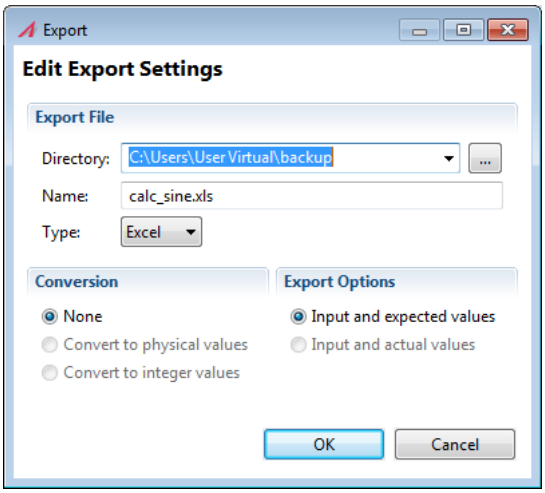


Figure 6.13: Export settings of data export

6.1.3 Properties view

Properties view

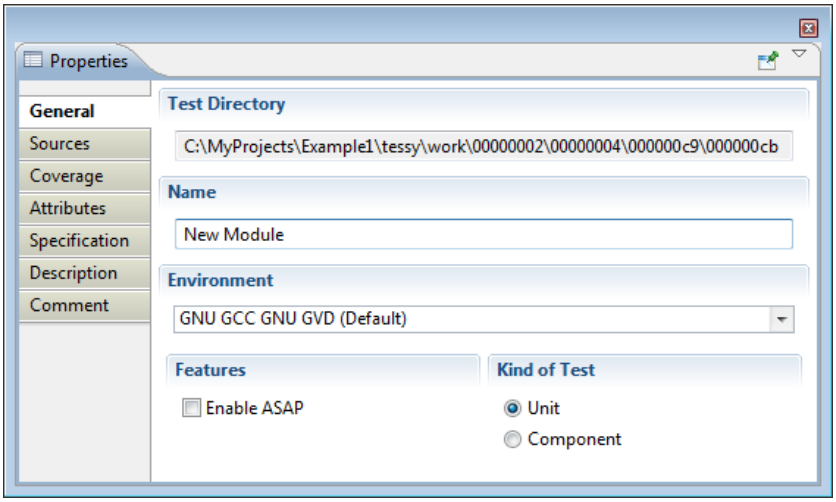


Figure 6.14: Properties view

The Properties view is divided into several tabs on the left and provides various settings explained in the following:

General

The General tab (see figure 6.14) is used to determine the test environment. Following options are available:

Option	Function
Test Directory	The path has been specified during database creation and is not adjustable here.
Name	Name of the element, e.g. test collection or module.
Environment	Specifies your target compiler (debugger/emulator/simulator) combination to be used for test execution. To enable the test environment see chapter 6.4 TEE: Configurating the test environment and section 6.4.6 Enable configurations. The GNU toolset is already available by default.
Kind of Test	Unit: Enables the unit test of TESSY. Component: Enables the component test of TESSY.
Features	"Enable ASAP": TESSY provides a close integration to the ASAP standard, allowing the usage of ASAP conversion rules for physical to integer conversion of test data. If ASAP is ticked, you will find additional attributes within the Attributes tab in which you have to specify your ASAP file. For more information refer to our application notes in the Help menu in the menu bar ("Help" > "Documentation" > "Using ASAP Information").

Table 6.27: General tab of Properties view

Sources: Sources pane

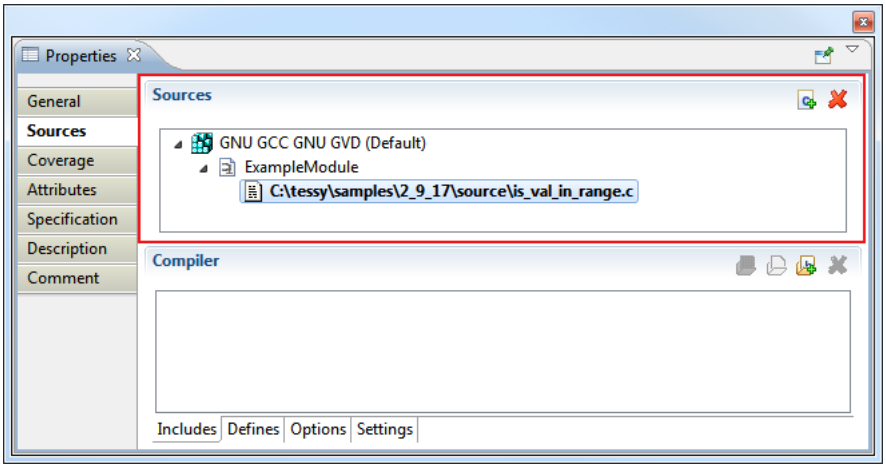


Figure 6.15: The Sources pane in the Sources tab of the Properties view

Adding the C-source file

On the Sources tab in the upper Sources pane you add the source files to be tested. All exported functions will be displayed if the module is opened. Some additional compiler options can be specified on module level by selecting the module entry, other options can be specified for each source file in the list.

→ Click on (Add Source).

→ Select your C-source file.

→ Click "Open". The C-source file will be added.

To remove a source file:

→ Select a source file and "Remove File" from the context menu.

To replace a source file:

→ Select a source file and "Replace File" from the context menu.

→ From the next dialog, select another source file.

Sources: Compiler pane

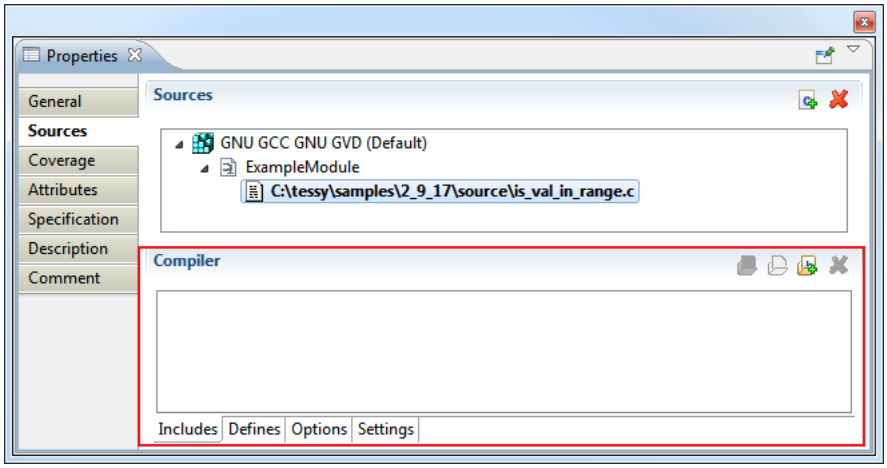


Figure 6.16: The Compiler pane in the Sources tab of the Properties view

The lower Compiler pane displays information about the item selected from the upper Source pane. Some of the displayed options (e.g. Includes) in the lower Compiler pane can be specified in the Test Environment Editor and will be inherited from there.

What kind of information is visible depends on the tab Includes, Defines, Options, Settings which you have selected. You may specify additional settings in each of the provided tab. Module options apply to all source files unless otherwise specified on file level. File options apply to one selected source file and will overwrite options that are specified on module level.

Options of the Compiler pane:

Includes

To add an include path: *Including path*

→ Click on (Add Include Path).

→ Select an appropriate path.

→ The path will be inserted.

Defines

If necessary, define a macro for the preprocessor:

→ Click on (New Define).

→ Enter the name of the define without the normally used option of your target compiler, e.g. -D. TESSY will use the appropriate option automatically.

Defines

To edit defines:

→ Click on (Edit Defines).

The Edit Defines As Text dialog will open.

Macros have to be separated by a comma or semicolon.

Options

Specify additional directives for your target compiler for your needs. Note that macros for the preprocessor and include paths have to be specified within the Defines tab respectively within the Includes tab.
All compiler options added here will be used for the compilation of the source file when building the test driver application.

Settings - module selected

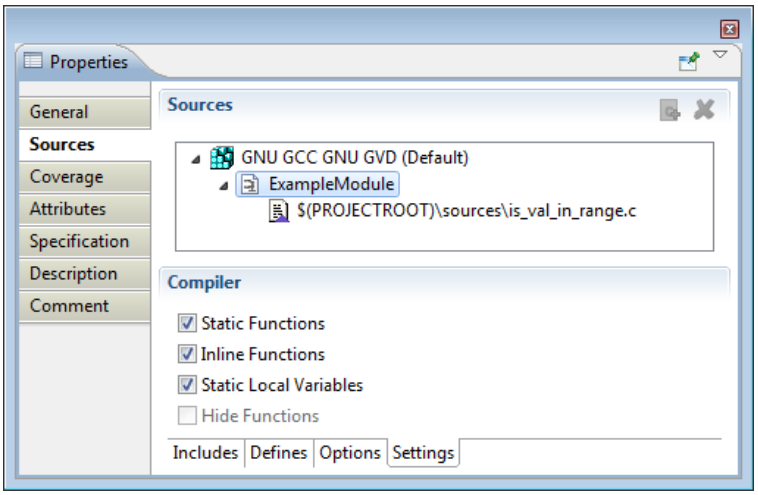


Figure 6.17: The Setting tab of the Properties view with module selected

If the module is selected, you can force TESSY to enable static functions, static local variables or to hide inline functions. These options apply to the whole module though it may be selected only when the module entry is selected.

Static Functions	If this box is not checked, no static function will be listed in the test object list of the module.
Inline Functions	If this box is not checked, no function with the keyword "inline" will be listed in the test object list of the module.
Static Local Variables	Static local variables within functions may be used as normal input or output variables within TESSY. The interface parser will recognize those variables and they may be set and evaluated.

Table 6.34: Options of module settings in Properties view

Settings - C-source file selected

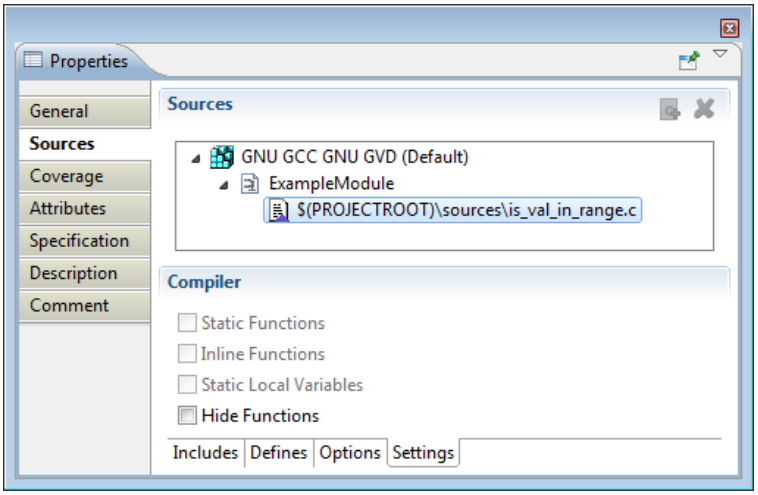


Figure 6.18: The Setting tab of the Properties view with C-source file selected

If the C-source file is selected, you can force TESSY to hide the functions of the C-source file from being displayed in the list of test objects.
This option is useful for additional C-source files needed for testing (e.g. implementation of stub functions for called functions), since they are not relevant for testing and reporting.

Coverage

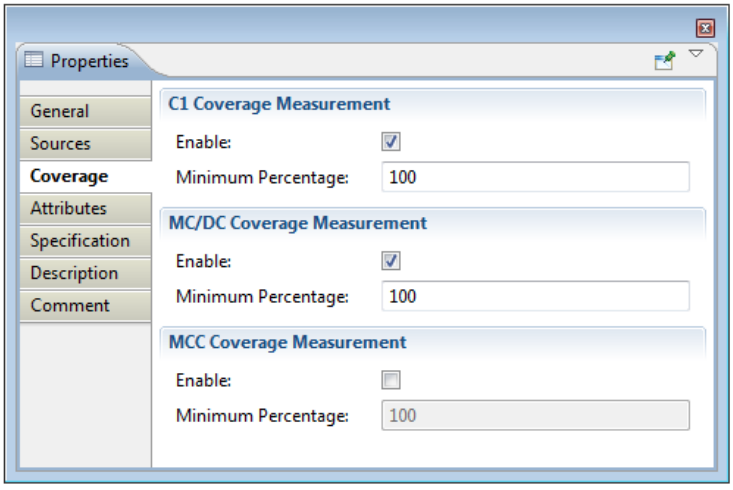


Figure 6.19: The Coverage tab of the Properties view

Within the Coverage tab you can enable the various coverage measurements.

see 5.1.9 Repeating the test run with coverage instrumentation

The C1 and MC/DC coverage are enabled by default. The default minimum coverage is 100%. You may lower the minimum coverage for individual test objects if there are unreachable branches or conditions within those test objects. This allows to you get succesful coverage results even for unreachable branches/conditions.

Attributes

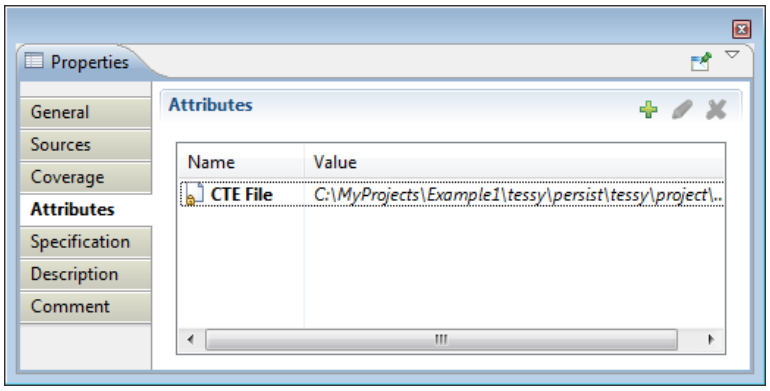


Figure 6.20: The Attributes tab of the Properties view

The Attributes tab specifies settings required by the compiler or the target environment of the module. Most attributes were preset and inherited from the Test Environment Editor (TEE).

You can change the default values or add new attributes to the Attributes pane:

Insert attributes

Changes are carried out only locally and do not influence other modules.

To create a new attribute:

→ Click on (New Attribute).

The Edit Attribute Properties dialog will open.

→ Enter an attribute name and select an appropriate type, e.g. String. Available types are String, Boolean, Integer, Real, File, Folder and Url.

→ Select appropriate flags, depending on the type selected.

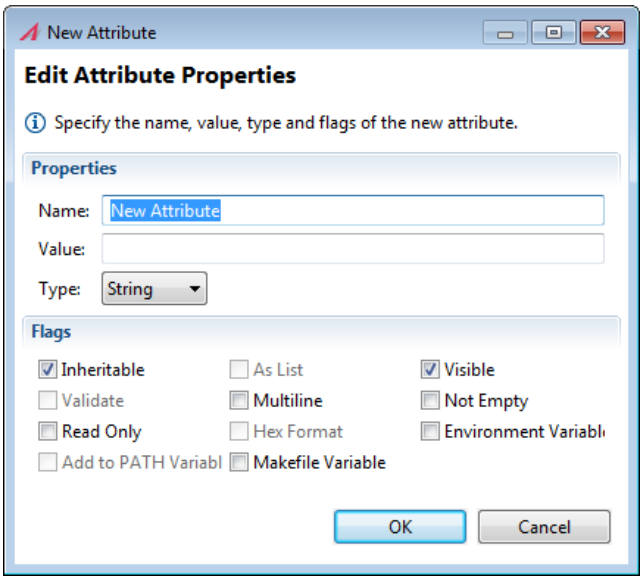


Figure 6.21: Creating a new attribute

To edit an existing attribute:

→ Click on (Edit Attribute).

The Edit Attribute Properties dialog will open.

You can remove user defined attributes. You cannot remove default attributes, only reset the value to its default state, if changed before.

To remove an attribute respectively reset a default attribute:

→ Click on .

Specification / Description / Comment

Those tabs provide editable textboxes to be used for specifications, descriptions and comments by the tester.

6.1.4 Requirements Coverage view

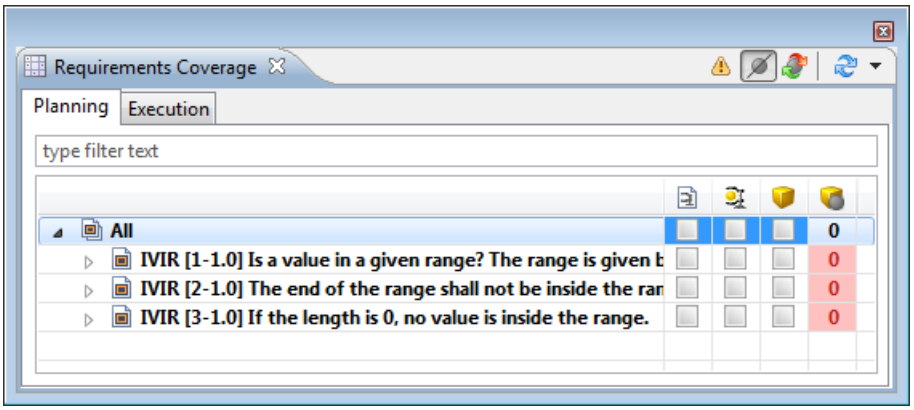

Requirements Coverage
Planning
Execution
type filter text
All
IVIR [1-1.0] Is a value in a given range? The range is given b
IVIR [2-1.0] The end of the range shall not be inside the ran
IVIR [3-1.0] If the length is 0, no value is inside the range.
0
0
0
0

Figure 6.22: Requirements Coverage view

Within the Requirements Coverage view you can link the requirements with your test cases. We will describe this view in section 6.3 Requirement management > 6.3.12 Requirements Coverage view.

6.1.5 Test Items view

Test Items view

In the Test Items view you get an overview about your test cases and test steps, and you can as well create test cases and test steps manually without using the Classification Tree Editor (CTE, see section 6.6). This is useful for simple test objects with a few test cases that can be documented in a few words manually.

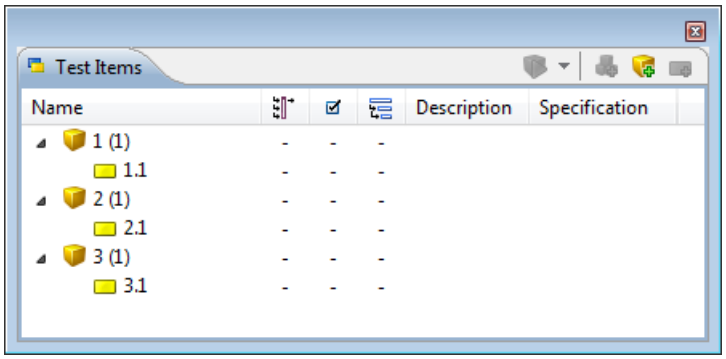


Figure 6.23: Test Items view

Icons of the view tool bar

icon	action / shortcut / comment
	Displays expected results.
	Displays evaluation macros results.
	Displays call trace results.
	Executes selected test cases.
	Adds new test case.
	Adds new test step.

Table 6.38: Tool bar icons of the Test Items view

Status indicators

indicator	status / meaning
	The test case has no data.
	The test case contains any data.
	At least one test step of the test case is ready to be executed.
	The test step has no data.
	The test step contains any data.
	The test step is ready to be executed.
	The test case has been created by the CTE and therefore can be changed only within CTE. The test case does not contain any data.
	The test case has been created by the CTE and therefore can be changed only within CTE. The test case does contain some data.
	The test case has been created by the CTE and therefore can be changed only within CTE. At least one test step is ready to be executed.

continue next page

indicator	status / meaning
	The test step has been created by the CTE and therefore can be changed only within CTE. It does not contain any data.
	The test step has been created by the CTE and therefore can be changed only within CTE. It does contain some data.
	The test case has been created by the CTE and therefore can be changed only within CTE. At least one test step is ready to be executed.

Table 6.39: Status indicators of the Test Items view

Creating test cases and test steps

To create test cases and test steps

Creating test cases

→ Switch to the Test Items view.

→ Click on (New Test Case).

The first test case is created and a test step is automatically added.

→ Add further test steps with a click on (New Test Step)

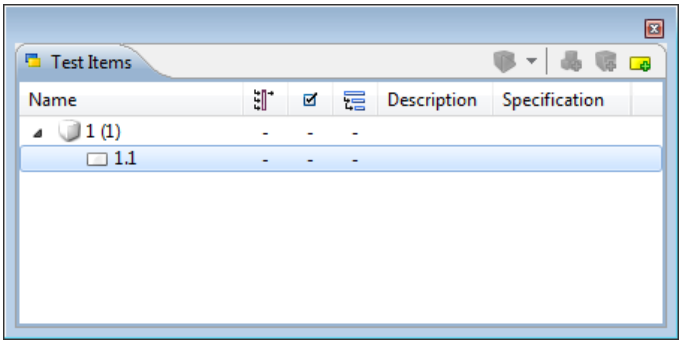


Figure 6.24: First test case with one test step.

Please notice the following habits of this view:

- The first number is the number of the test case, the number in brackets shows the quantity of the test steps included.
- Test case numbers will be counted continuously: If you delete test cases, new test cases will get a new number and existing test cases will not be renumbered.
- If you cannot click on "New Test Case" oder "New Test Step" because the icons are inactive, you might be in the wrong selection: Select the test object within the Test Project view, then select the Test Items view.
- If you double click a test case, the TDE will be openened to enter test data. Make sure to adjust or review the passing directions first in the TIE.

Every test step contains a complete set of test data. For instance, the mechanism of test steps can be used to achieve an initialization of the test object before executing the test step that checks the actual test condition of the current test case.

Important: If test cases and test steps were assigned within CTE, the icons of test cases and test steps within the Test Items view are displayed with a CTE symbol to indicate you can change those test cases only within CTE (see figure 6.25).

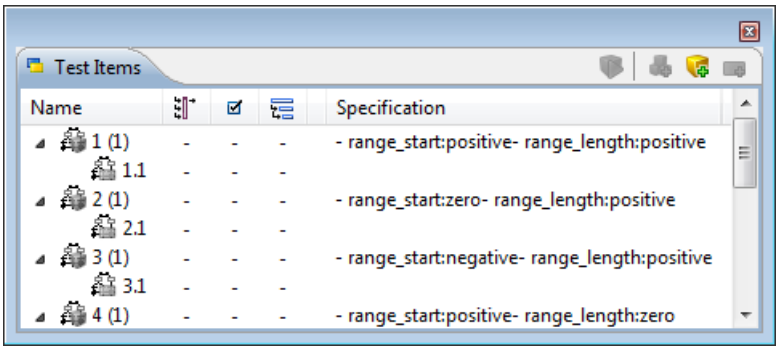

Test Items
Name
Specification
1 (1) - - - - range_start:positive- range_length:positive
1.1 - - -
2 (1) - - - - range_start:zero- range_length:positive
2.1 - - -
3 (1) - - - - range_start:negative- range_length:positive
3.1 - - -
4 (1) - - - - range_start:positive- range_length:zero

Figure 6.25: Test cases created by CTE

6.1.6 Test Results view

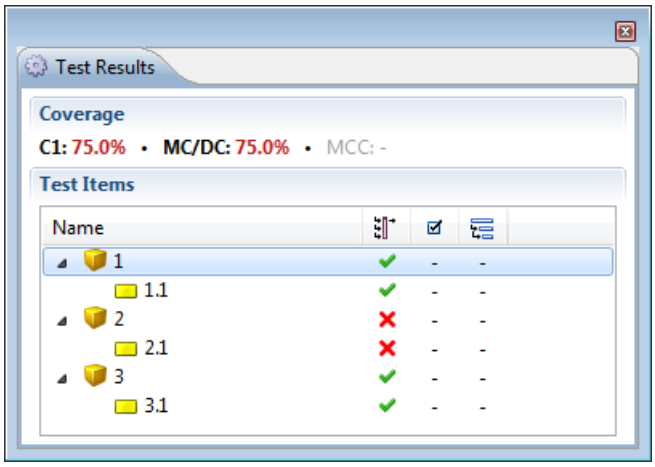


Figure 6.26: Test Results view

After a test run the Test Results view will display the coverage measurement results and the results of expected outputs, evaluation macros and call traces, if applicable.

Important: The view is context sensitive: If the Test Results view is empty, make sure a test run is selected within the Test Project view!

6.1.7 Evaluation Macros view

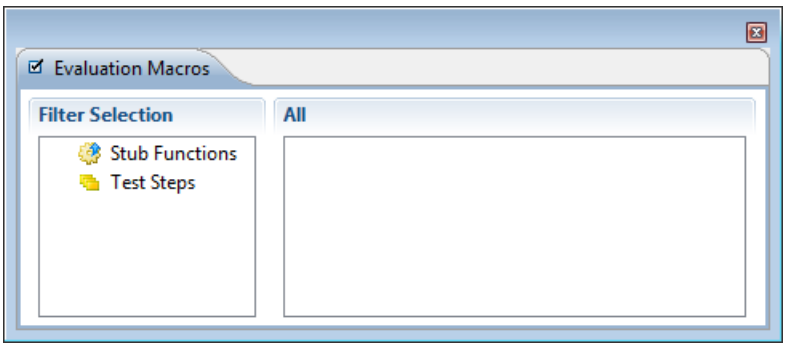


Figure 6.27: Evaluation Macros view

This view lists the detailed results of the evaluation macros if the usercode of the test object contains any evaluation macros, see 6.7.9 Using evaluation macros. The results

are displayed wherever they occur within the usercode, e.g. within stub functions or test step epilogs.

6.1.8 Console view

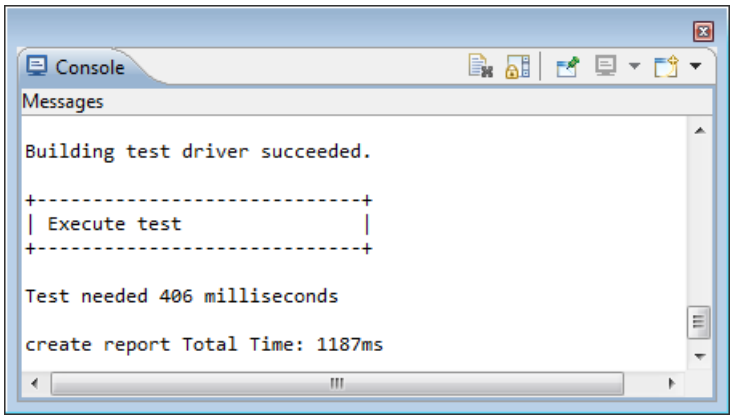


Figure 6.28: Console view

The Console view displays messages of sub processes invoked during the compilation and execution process of the test driver application. It provides a quick overview about any error messages.

You can enable the console view to be shown whenever an error occurs during C-source analysis or test driver compilation:

- → In the menu bar blick on "Window".
- → Open the "Preferences".
- → Click on "Test Execution Settings" and check the setting "Show console on error" (see figure 6.29.

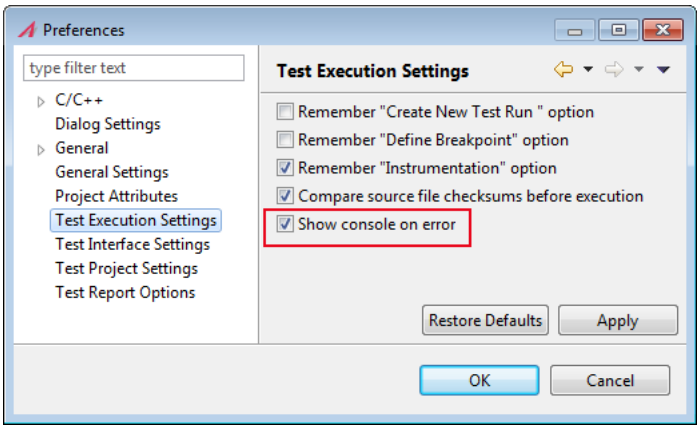


Figure 6.29: Preference "Show console on error"

6.1.9 Suspicious Elements view

Since the view refers to changes of requirements, we will discuss this view in section 6.3.7 Suspicious Elements view.

6.2 Import perspective: Importing requirements

The Import perspective is not displayed within the perspective bar by default.
To open the Import perspective,

→ switch to the Requirement Management perspective.

→ In the tool bar above click on (Show Import Perspective).

The perspective "Import" will open.

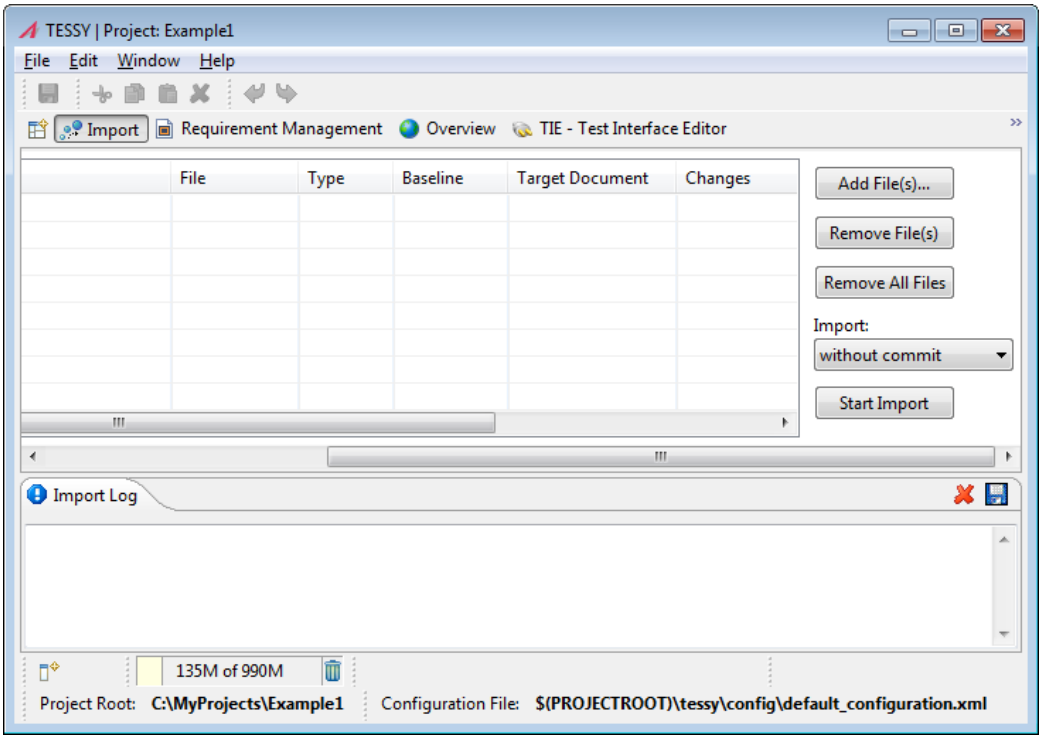


Figure 6.30: Import perspective

6.2.1 Importing requirements

Importing requirements

To import files, e.g. requirements:

→ Click on "Add File(s)".

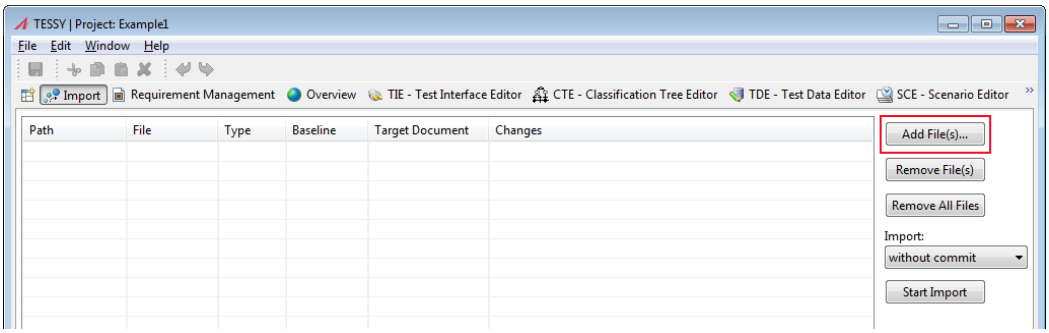


Figure 6.31: Adding files in the Import perspective

→ Select the files you want to import and click "Open".

→ Select the commit-option of the import, i.e. "without commit", "commit changes (abord on problem)" or "commit changes (ignore problems)".

→ Click on "Start Import".

You can choose which requirement document shall be created or updated with the imported data.

The import can be used with the following options that control the versioning of the imported requirements:

Option	**Comment**
Import without commit	You will see the changes made to your requirement document after the import operation. This allows rejecting the import and revert the document to the version that was available before the import operation.
Import with commit	This option causes an automatic check-in of all changes after a successful import operation. Any change to an existing requirement will cause the creation of a new requirement version.

Table 6.47: Committing options to control the versioning of imported requirements

Import Log view

The Import Log view in the lower center of the Import perspective monitors the status of the import.

Please notice the following habits:

- If no requirement document exists, the name of the import file will be used as document name.
- If only one document exists, the imported requirements will be used to update this document. You will see any changes resulting from the import within the requirements perspective (see section 6.3.10 History view).
- If there is more than one document available, you can select the desired target document where to import the new requirements.

6.2.2 Formats

You can import following formats of requirement sources:

Format	Comment
*.txt	Simple ASCII format were each line is recognized as a requirement. This is the very basic format that allows importing all sorts of text as requirements.
*.csv, *.tsv	Comma or tab separated ASCII text with a headline indicating the column names. This format allows specifying requirement id, version and all other available requirement properties.
*.xml	TESSY specific XML format which provides specifying the chapter structure of a requirement document. All available requirement properties may be specified within this format. It is the recommended exchange format when importing requirements from any other requirement management systems.

Table 6.50: Possible formats of requirement sources

6.3 Requirement management

The basis for all testing activities should be a precise functional specification of the system under test. All testing activities should be caused by requirements described within the functional specification and each change of the requirements need to be tracked in order adjust the tests, if necessary. That is the reason why TESSY incorporates a requirement management solution that provides a basic environment for requirements engineering with the following features:

- Exporting and importing requirements
- Creating new requirements
- Comparing of requirement versions
- Automatic versioning of requirement changes
- Adding images to requirements
- Marking different versions of requirements as semantically equivalent

The requirement management takes place in various perspectives and views:

1. You will import your requirements within the Import perspective.

6.2 Import perspective: Importing requirements

2. For creating requirements, tracking changes, versioning and history you will use the Requirement Management perspective.

6.3.1 Structure of the Requirement Management perspective

3. Linking requirements with test cases takes place within the Link Matrix view or the Requirements Coverage view of the Overview perspective.

6.3.12 Requirements Coverage view

6.3.1 Structure of the Requirement Management perspective

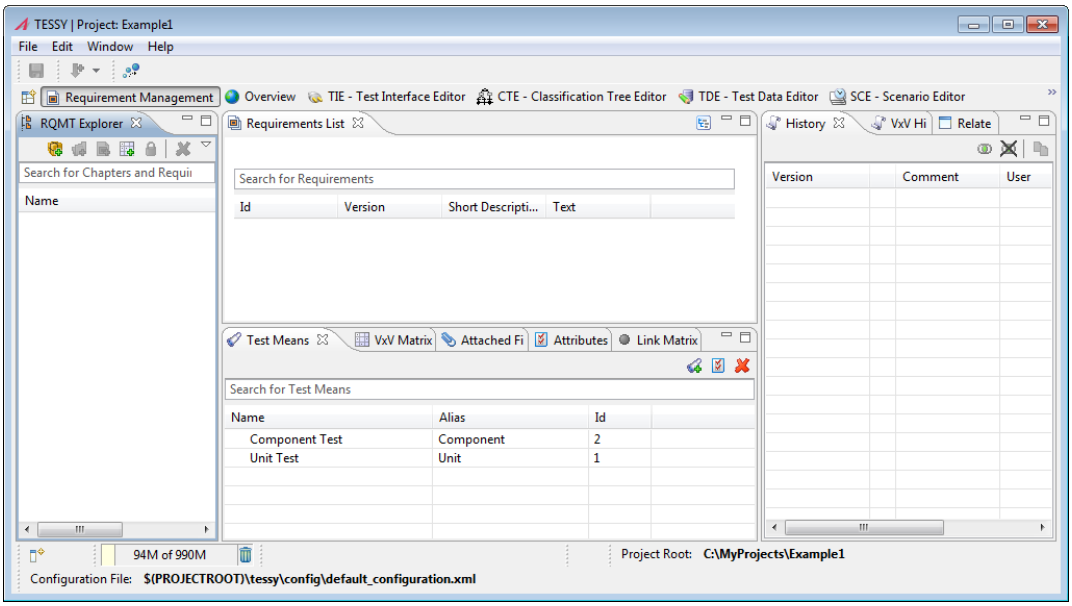


Figure 6.32: Requirement Management perspective

pane	**location** (default)	**function**
RQMT Explorer view	left	To create requirements.
Requirements List view	upper center	To view imported requirements as list for a selected document or folder.
Test Means view	lower center	To list the available test means, up to unit test and component test.
VxV Matrix view	lower center	To assign requirements to test means.
Link Matrix view	lower center	To link requirements with modules, test objects and test cases.
Attached File view	lower center	To attach files to requirements.
Attributes view	lower center	To edit a list of attributes for a requirement.
Suspicious Elements view	lower center	To have a quick look over all suspicious (modified) elements.

continue next page

pane	**location** (default)	**function**
History view	right	To display the version history of the selected requirement.
VxV History view	right	To display the version history of the verification matrix.
Related Elements view	right	To display linked elements for a selected requirement.
Requirements Coverage view	**Overview perspective!**	To select and link the requirements.

Table 6.52: Structure of the Requirement Management perspective

Important: To gather all information about managing requirements within this chapter, we will describe the Requirement Coverage view in this chapter, although the workflow makes it reasonable to place this view within the Overview perspective.

6.3.2 RQMT Explorer view

RQMT Explorer view

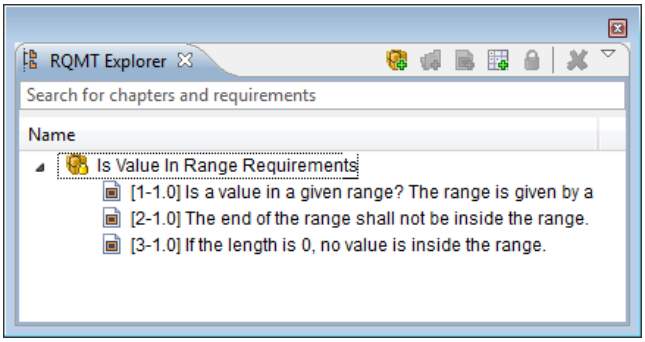

RQMT Explorer
Search for chapters and requirements
Name
Is Value In Range Requirements
[1-1.0] Is a value in a given range? The range is given by a
[2-1.0] The end of the range shall not be inside the range.
[3-1.0] If the length is 0, no value is inside the range.

Figure 6.33: RQMT Explorer view

Icons of the view tool bar

icon	action / shortcut / comment
	Creates a new document.
	Creates a new chapter.
	Creates a new requirement.
	Creates a new validation and verification matrix.
	Locks the marked item.
	Deletes the marked item.

Table 6.53: Tool bar icons of the RQMT Explorer view

Status indicators

indicator	status / meaning
	The requirement document is suspicious.

Table 6.54: Status indicators of the RQMT Explorer view

Renaming / Alias

You can rename a requirement document and give an alias. This is useful for the report, in order to have an abbreviation of the document name when building the requirement identifier.

To give an alias:

→ Select the document and "Rename" from the context menu.

→ Enter an Alias.

→ Click "OK".

The requirement document will be displayed as "modified" (see figure 6.34).

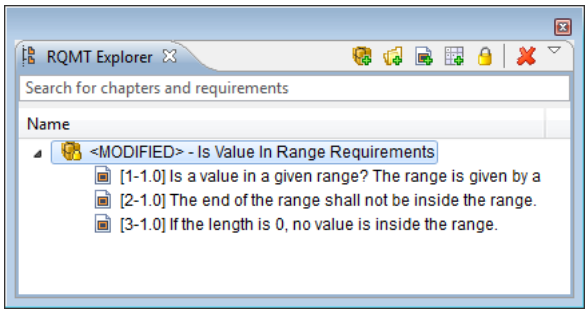

RQMT Explorer
Search for chapters and requirements
Name
<MODIFIED> - Is Value In Range Requirements
[1-1.0] Is a value in a given range? The range is given by a
[2-1.0] The end of the range shall not be inside the range.
[3-1.0] If the length is 0, no value is inside the range.

Figure 6.34: The document is modified, because the alias was modified

Committing requirements

Committing requirements

The requirements document needs to be checked-in as initial revision:

→ Select the document within the RQMT Explorer view and click on (Commit Changes) in the global tool bar.

You can commit all documents or selected elements (see figure 6.35).

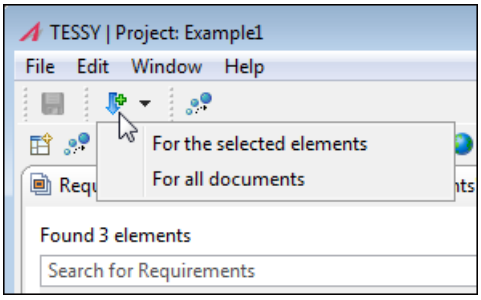


Figure 6.35: Committing options

→ Enter a comment and click "OK" (see figure 6.36).

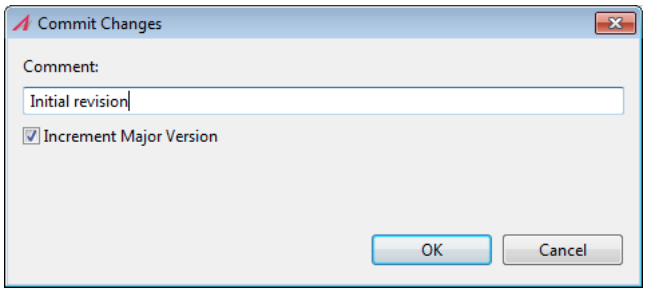


Figure 6.36: Comment for the initial revision of the commit

An initial revision of the requirement document will be created.

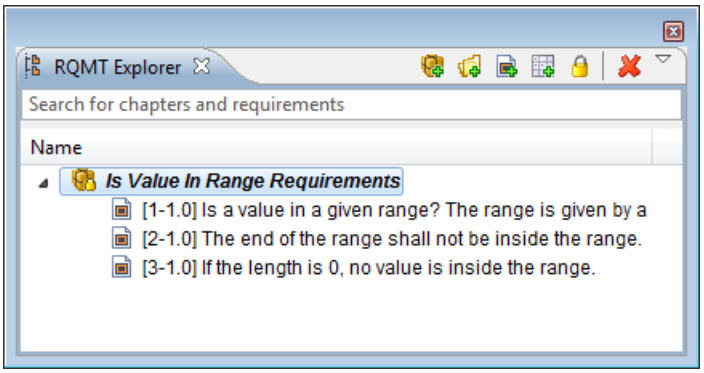

RQMT Explorer
Search for chapters and requirements
Name
Is Value In Range Requirements
[1-1.0] Is a value in a given range? The range is given by a
[2-1.0] The end of the range shall not be inside the range.
[3-1.0] If the length is 0, no value is inside the range.

Figure 6.37: After the commit

Creating requirements

Creating requirements

To create requirements:

→ In the RQMT Explorer view tool bar click on (New Requirement Document).

→ Click on (New Chapter). Chapters are optional to structure a requirement document.

→ Click on (New Requirement) to create a requirement.

Editing requirements

Editing requirements

To edit requirements:

→ Double click within the RQMT Explorer view on the requirement you want to edit.

The requirement will be opened within the requirement editor in the center of the Requirement Management perspective.

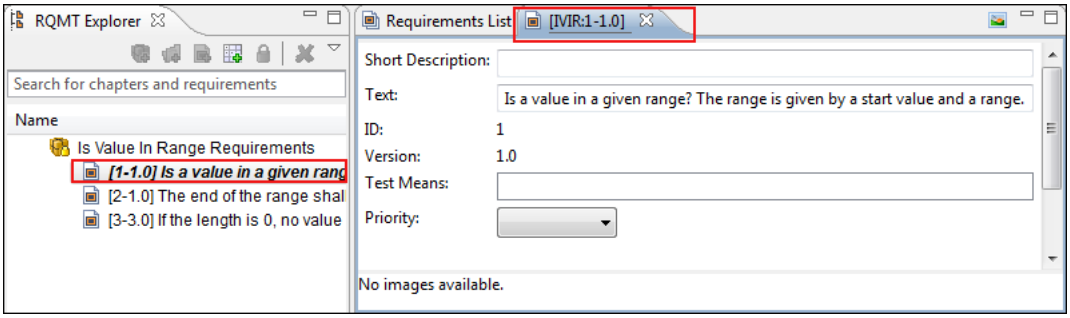


Figure 6.38: Double click on the RQMT opens the requirement editor

→ After editing, click on to save the changes.

TESSY will create now a locally modified version of the requirement which will be illustrated with the term *MODIFIED* in front of the requirement name (see figure 6.39).

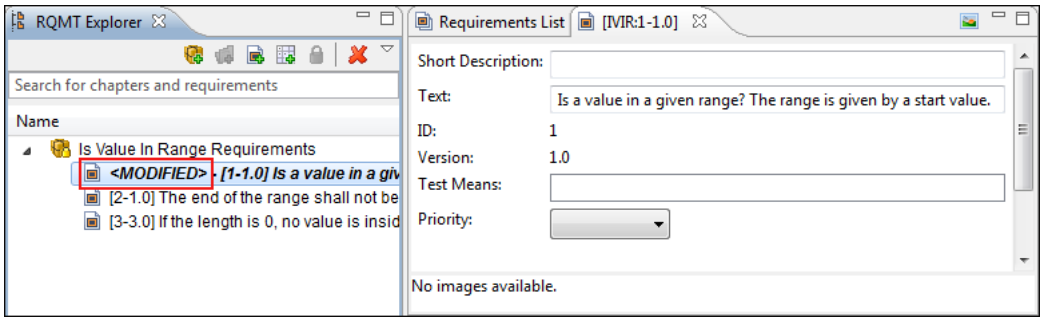


Figure 6.39: The first requirement was modified

→ Commit your changes with a click on .

→ A new version of the requirement will be created and you need to decide either to increment the major or the minor version number within the check-in dialog.

If you did only minor changes or want to commit a draft update of a requirement, you can decide to increment only the minor version. In all other cases, it is recommended to increment the major version.

After committing, the ID of the requirement changes to display the version (see figure 6.40).

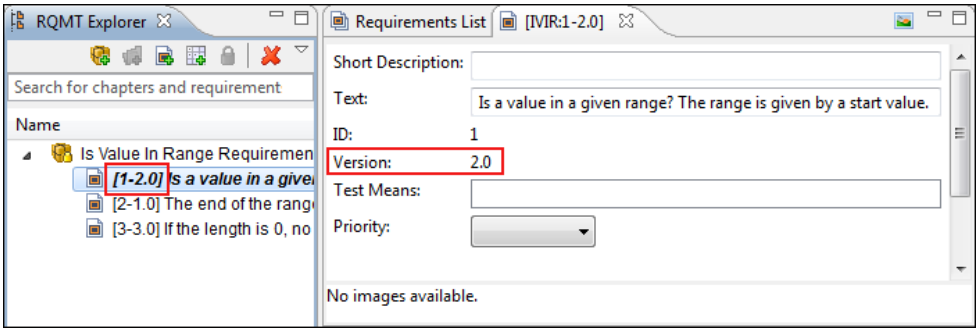


Figure 6.40: The first requirement has the version 2.0

6.3.3 Requirements List view

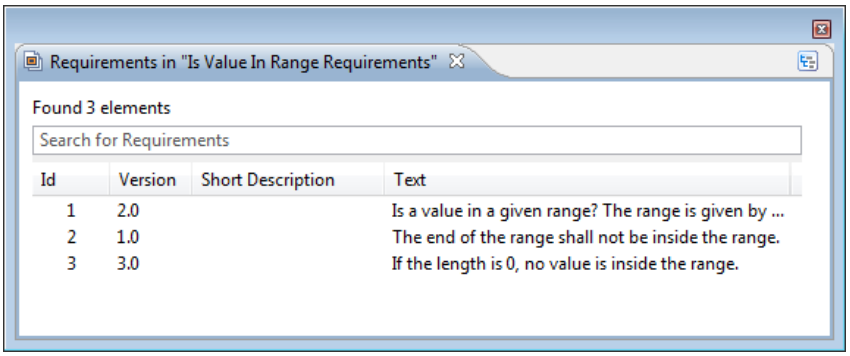


Figure 6.41: Requirements List view

Icons of the view tool bar

icon	action / shortcut / comment
	Shows all descendants.

Table 6.63: Tool bar icons of the Requirements List view

Viewing requirements, versions, IDs

Within the Requirements List view the requirements are displayed with the text, figures, if available, versions and IDs (see figure 6.42). *Requirements List view*

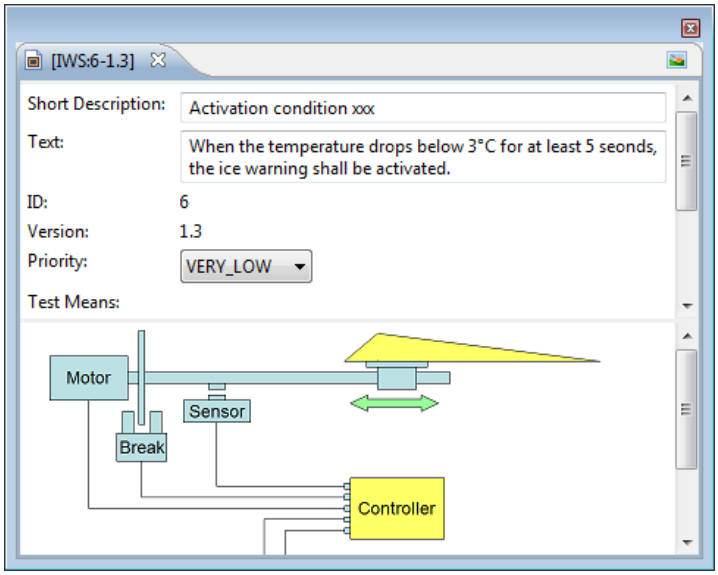


Figure 6.42: Requirements List view with test and a figure

Every requirement has an explicit ID. TESSY provides the following two mechanisms for assigning requirement version IDs:

- **Automatic**: TESSY assigns and increments version IDs automatically if the import file does not contain version numbers. This is the normal behavior when importing requirements from text files or spreadsheets. When checking-in requirements, you can decide to increment the major or minor version number (e.g. major 1.0 to 2.0 or minor 1.0 to 1.1).
- **Using external IDs**: It is possible to use existing requirement version numbers from the import file. This is useful if you get requirements from an external requirements management tool and need to import exactly the same version numbers that where already assigned to the requirements. The only prerequisite for this kind of import is the consistency of the version numbers.

When using external version IDs, the following checks of the imported data will be performed when importing: *Using external version IDs*

- If any requirement content is changed, the external ID must have been incremented. Otherwise an error will be reported for the respective requirements.

- If the version ID was changed but no requirement content was changed, a warning will be reported.
- If the new version ID is less than the highest existing ID for a requirement, an error will be reported.

Gaps within the numbering of requirement IDs are allowed.
The numbering has to be ascending.

6.3.4 Test Means view

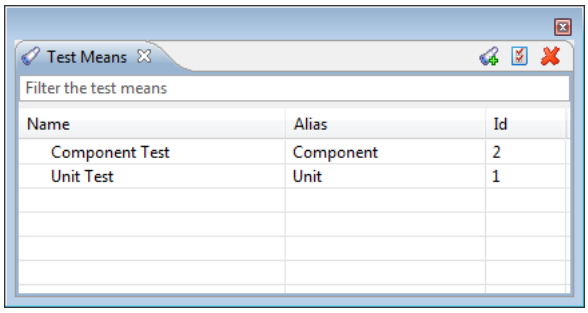


Figure 6.43: Test Means view

Requirements will be tested using different test means, e.g. unit test, system test or review. The default test means used within TESSY are for unit and component testing. You can filter your requirements by test mean for later reporting issues.

Icons of the view tool bar

icon	action / shortcut / comment
	Creates a new test mean.
	To edit the selected test mean.
	Deletes the selected test mean (Del).

Table 6.66: Tool bar icons of the Test Means view

6.3.5 VxV Matrix view

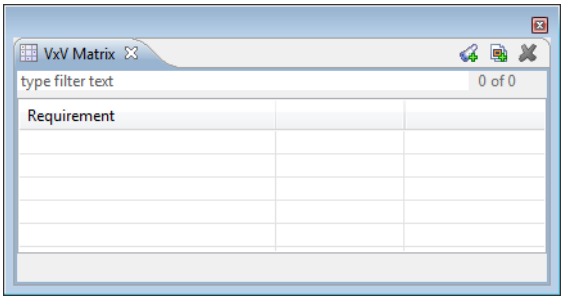


Figure 6.44: VxV Matrix view

The VxV matrix supports the assignment of requirements to the test means used for validation of the requirement. This helps filtering out those requirements that are to be tested with unit and component testing. The assignments within the VxV matrix will be used for requirement filtering for reporting.

Icons of the view tool bar

icon	action / shortcut / comment
	Adds all test means.
	Adds all requirements.
	Removes all unused items.

Table 6.67: Tool bar icons of the VxV Matrix view

6.3.6 Link Matrix view

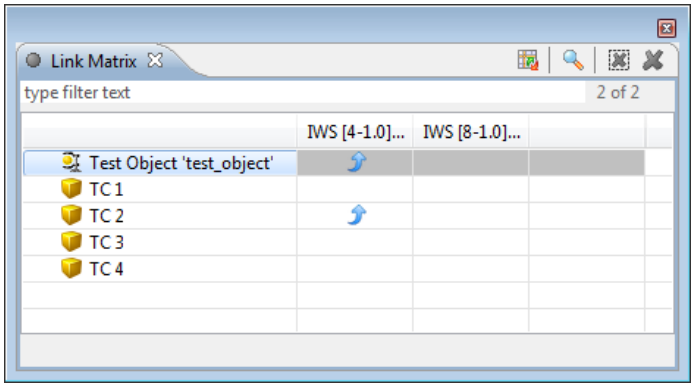


Figure 6.45: Link Matrix view

Link Matrix view

Within the Link Matrix view you can link modules, test objects and test cases with requirements. It shows the link relationship of the elements currently contained within the matrix.

Icons of the view tool bar

icon	action / shortcut / comment
	Transposes the matrix, e.g. changes the rows and columns (Ctrl+T).
	Adds all currently linked elements.
	Removes selected elements (Del). Does NOT delete the element but removes it from the view.
	Removes all elements. Does NOT delete the element.

Table 6.68: Tool bar icons of the Link Matrix view

Status indicators

indicator	status / meaning
	The link is suspicious.

Table 6.69: Status indicators of the Suspicious Elements view

Adding and removing elements

To add elements to the Link Matrix view:

→ Drag and drop requirements, modules, test objects or test cases into the matrix. The elements will be shown either within the rows or columns of the matrix.

> To exchange rows or colums, click on within the view tool bar.

→ Use the context menu entry "Add to Link Matrix" within the requirements explorer, Test Project view or Test Items view (see figure 6.46).

To remove elements from the Link Matrix view:

→ Click on (Remove All Elements) to remove all currently displayed elements.

> This will only remove the elements from the matrix view, no changes will be made to the elements themselves.

→ Click on (Remove Selected Element) will remove the currently selected element within a row of the matrix (if any element is selected).

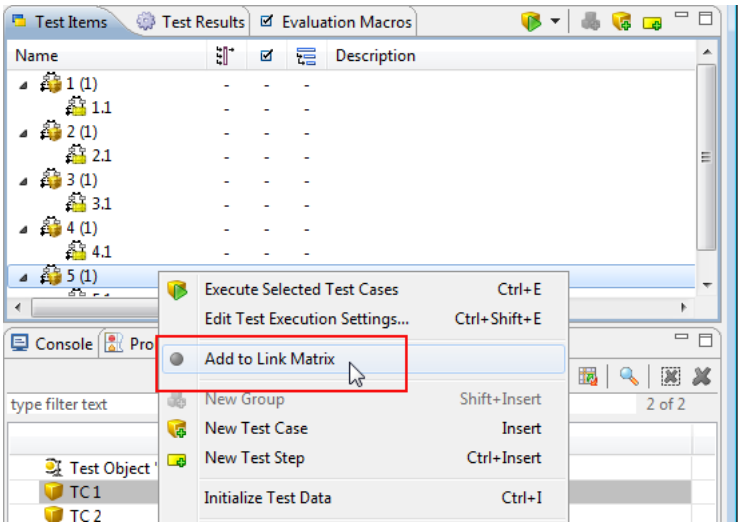


Figure 6.46: Adding Test Cases to Link Matrix view

Please notice the following habits:

- The current contents of the link matrix are remembered when restarting TESSY but the matrix itself is not persisted in any way. You can add or remove elements and this will not cause any changes to the elements.
- The search button "Add All Elements Linked to Elements in Rows" allows finding and adding the elements that are linked to the elements currently displayed within the rows of the matrix.
- The view is available within the Overview and within the Requirement Management perspective. Setting links or changing elements will cause dependent elements to become suspicious. Refer to the Suspicious view for details.

Updating requirement links

If requirements have changed, the links within the Link Matrix view will be declared suspicious with an exclamation mark (see figure 6.47).

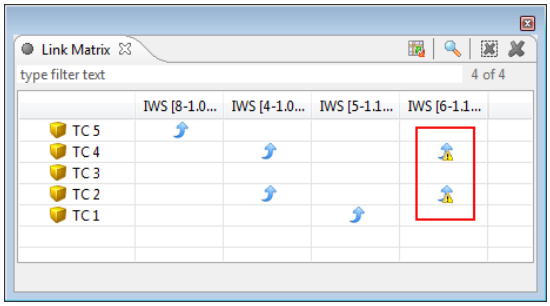


Figure 6.47: Link Matrix view with suspicious elements

Updating suspicious links

To update suspicious links:

→ Double click the link within the matrix. Another double click will delete the link and one more double click will create the link again.

→ Right click a cell and select "Update Selected Suspicious Link" from the context menu.

→ Right click a row and select "Update Suspicious Links" from the context menu. All links within the selected row will be updated.

6.3.7 Suspicious Elements view

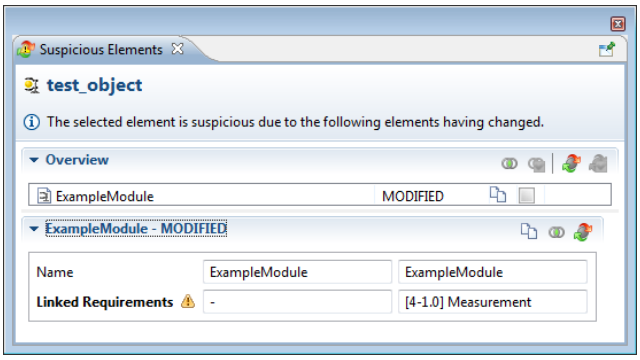


Figure 6.48: Suspicious Elements view

Suspicious Elements view

The Suspicious Elements view allows finding out the reason why an element is suspicious.

Suspicious elements

During the testing process modules, test objects and test cases will be linked to requirements indicating that the respective requirements are tested by the linked elements. Whenever a requirement changes because of modifications or because a new version has been checked in, the linked elements will become suspicious and need to be reviewed. The suspicious status will be indicated by an exclamation mark icon decorator, i.e. for a suspicious test object.

Icons of the view tool bar

icon	action / shortcut / comment
	Sets elements semantic equal (all elements or selected element).
	Updates links (all links or selected link).
	Compares the versions.

Table 6.75: Tool bar icons of the Suspicious Elements view

Determine changes that caused suspicious status

When you have linked the test object and some test cases, any changes to the linked requirements will cause the linked elements to become suspicious.

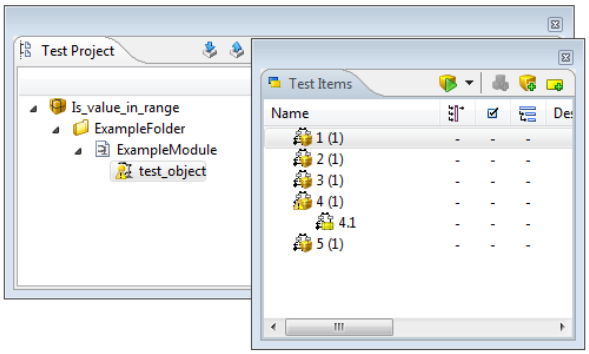


Figure 6.49: Suspicious test object and test cases

Determine the related modified requirements that causes the status of a test object being suspicious within the Suspicious Elements view:

→ Select the suspicious test object within the Test Project view (i.e. within the Overview perspective).

The Suspicious Elements view will display the changed requirements (see figure 6.50).

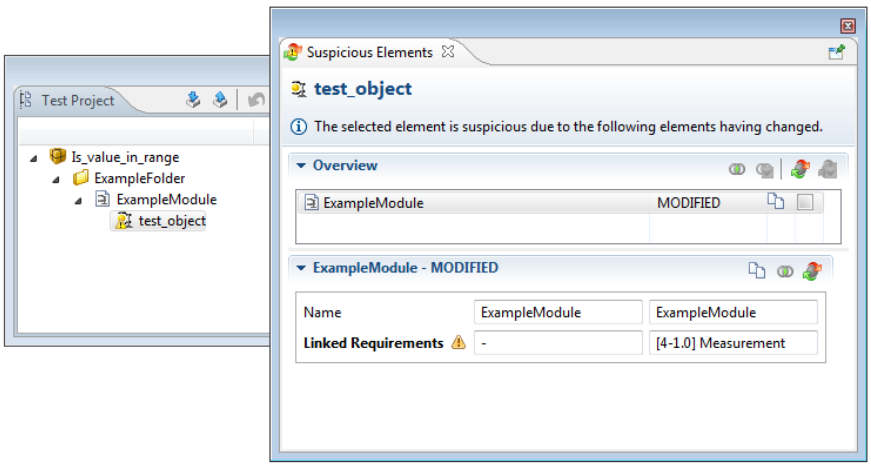


Figure 6.50: Suspicious test object and linked modified requirements

As you can see in figure 6.50 above, one requirement has been added to the module. If you select a test case, the Suspicious Elements view will show the changed requirement(s) (see figure 6.51).

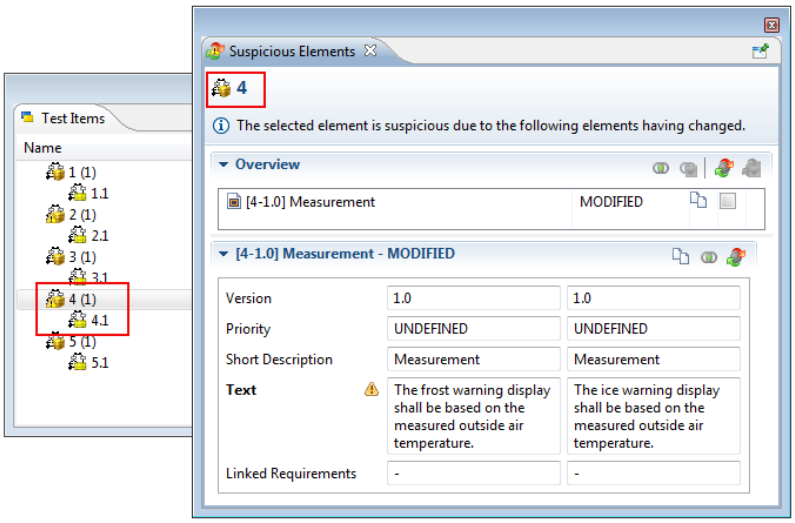


Figure 6.51: Selecting the suspicious test case shows the modified requirement(s)

Within the Differences view you can determine the exact differences: *Differences view*

→ Click on (Compare Versions).

The Differences view shows all changes of the requirement (see figure 6.52).

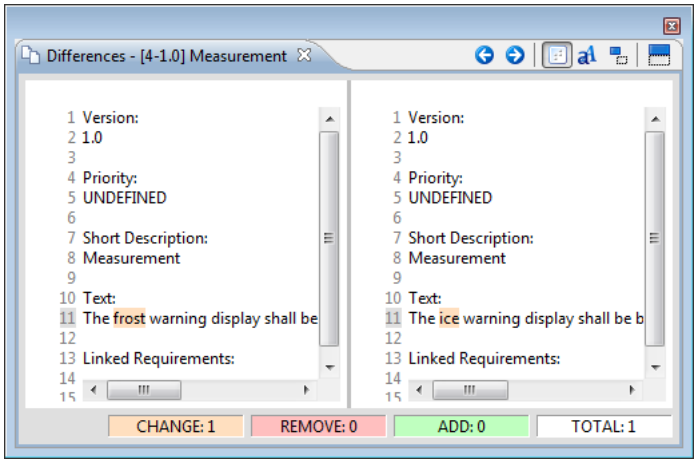


Figure 6.52: Comparing the versions of the requirement

You need to determine if the change of the requirement affects the linked test cases and adapt the test data if necessary. If no changes to the test cases are required, update the link to acknowledge the requirement change. The suspicious icon will then disappear for the respective test case. You can also use the Link Matrix view to update requirement links.

6.3.8 Attached Files view

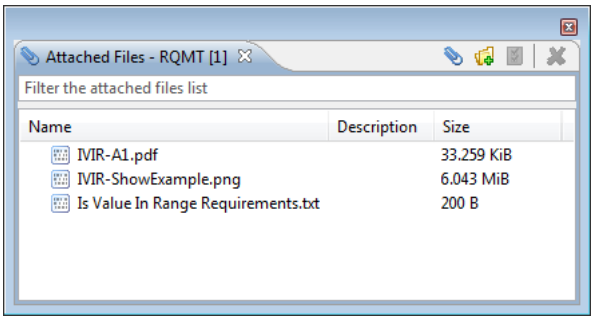


Figure 6.53: Attached Files view

Attached Files view

The Attached Files view allows adding arbitrary files to the selected requirement. You can add additional documents with detailed information about the requirement. The files will be stored within the TESSY database.

Icons of the view tool bar

icon	action / shortcut / comment
	Adds the selected file (Ctrl+N).
	Creates a new folder (Ctrl+B).
	To edit a name or description of the selected file or folder (Alt+C).
	Deletes the selected file (Del).

Table 6.78: Tool bar icons of the Attached Files view

6.3.9 Attributes view

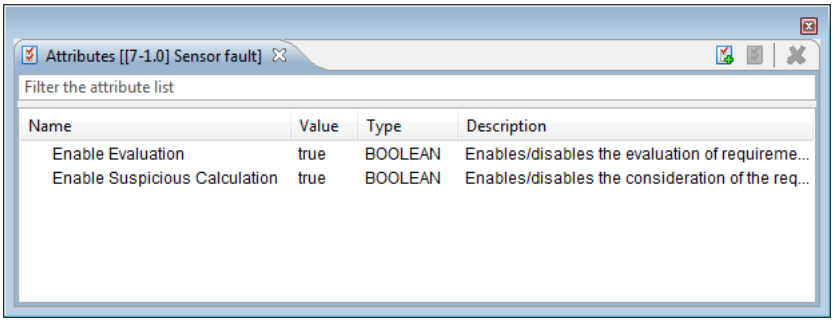


Figure 6.54: Attributes view

The Attributes view allows adding arbitrary attributes for the selected requirement or requirement document. *Attributes view*

Important: New attributes should be created for the requirement document. They will then be inherited to each requirement of the document and can be overwritten on requirement level.

There are some predefined attributes on document level that control the behavior of the requirement evaluation and suspicious calculation for elements linked to requirements.

Icons of the view tool bar

icon	action / shortcut / comment
	Creates a new attribute (Ctrl+N).
	Edits the selected attribute (Ctrl+E or double click).
	Deletes the selected attribute (Del).

Table 6.79: Tool bar icons of the Attributes view

Attributes appear in different font characters:

font character	means
bold	newly created
italic	overwritten for the selected requirement or document
normal	inherited attribute

Table 6.80: Tool bar icons of the Attributes view

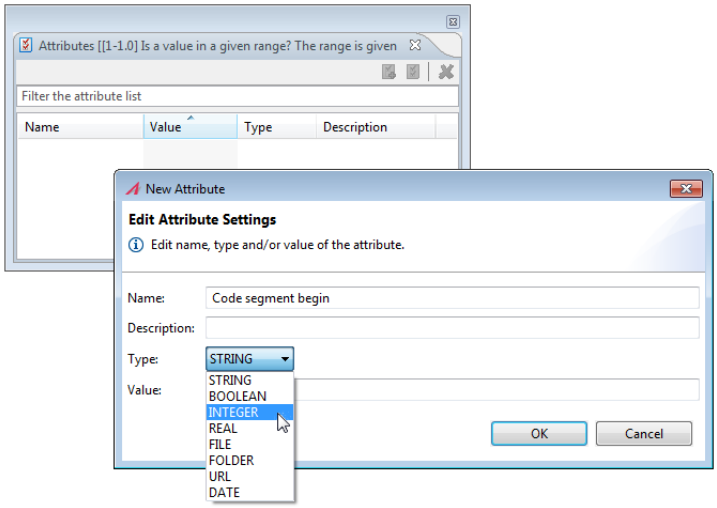


Figure 6.55: Editing attribute settings within the Attributes view

6.3.10 History view

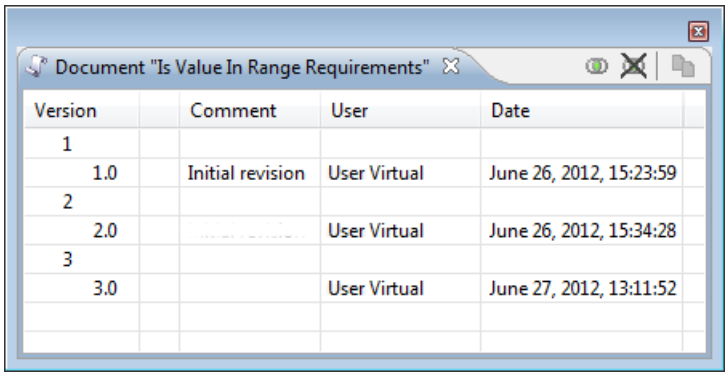


Figure 6.56: History view

The History view shows the version history of the currently selected element, e.g. requirement or document. *History view*

Icons of the view tool bar

icon	**action / shortcut / comment**
	Two versions of a requirement are set to be semantically equal if their contents reflect the same semantics but minor changes were made, e.g. spelling errors.
	Unsets semantic equal. You need to have two versions of a requirement selected to do this operation.
	Compares the versions.

Table 6.81: Tool bar icons of the Histroy view

Reviewing changes

Each requirement has a version history showing all of its changes. *Reviewing changes*
To review the changes between any two versions of the history or between a historic version and the current version,

→ select either two versions within the history view to compare these versions or select only one version within the view if you want to compare it against the current version.

→ Click on (Compare) in the toolbar.

Differences view

The Differences view will be displayed within the lower pane, which provides a direct comparison of the respective requirement versions printed as text (see figure 6.57).

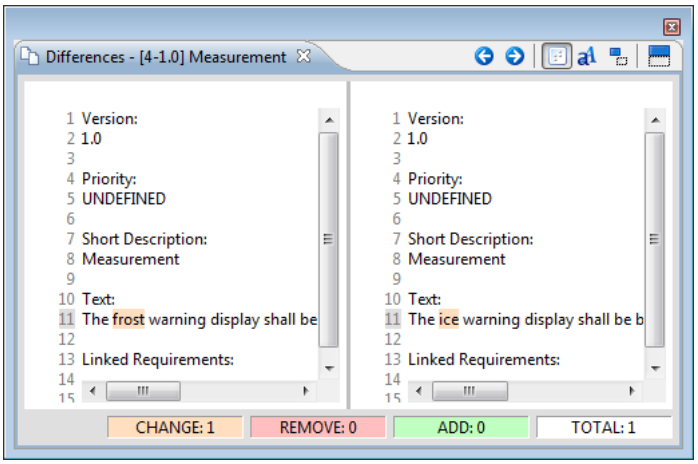


Figure 6.57: Differences view

6.3.11 Related Elements view

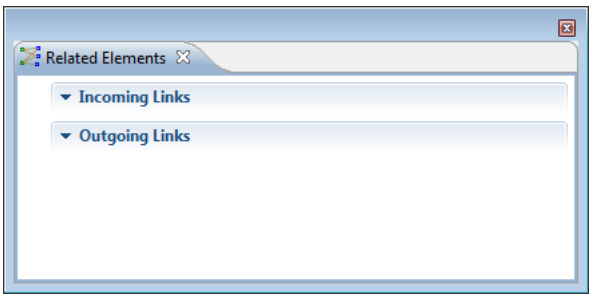


Figure 6.58: Related Elements view

In this view you can see the links of requirements to other requirements, e.g. when creating refined requirements based on a given requirements document. This view presents all linked elements of a requirement, e.g. all sub requirements or the linked main requirements.

6.3.12 Requirements Coverage view

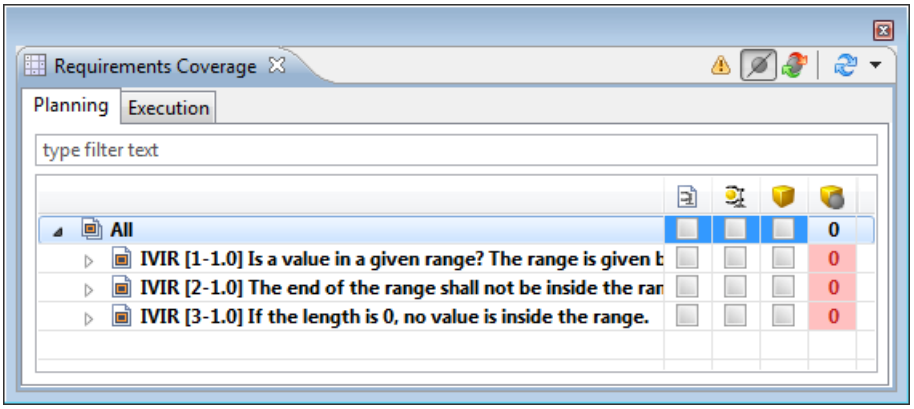

Requirements Coverage
Planning
Execution
type filter text
All
0
IVIR [1-1.0] Is a value in a given range? The range is given b
IVIR [2-1.0] The end of the range shall not be inside the ran
IVIR [3-1.0] If the length is 0, no value is inside the range.

Figure 6.59: Requirements Coverage view

Important: By default you will find the Requirements Coverage view within the Overview perspective!

Within the Requirements Coverage view you will link the test cases with the requirements. You will as well have an overview of the requirements coverage, therefore you will find this view within the Overview perspective.

Requirements Coverage view within the Overview perspective

Icons of the view tool bar

icon	action / shortcut / comment
	Shows all requirements, including unlinked requirements.
	Updates all links.
	Refreshes the view. With click on the little arrow next to the icon, you can choose on which selection you want to refresh (see figure 6.60).

Table 6.83: Tool bar icons of the Requirements Coverage view

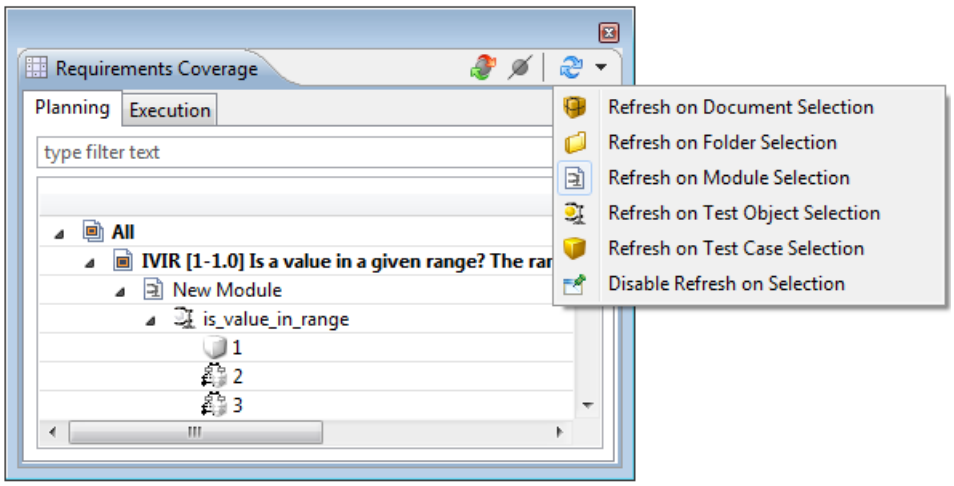


Figure 6.60: Options of refreshing

Linking requirements with test cases

The idea behind linking requirements to modules and test objects is the following process:

- First, the complete list of requirements is gathered.
- Second, each applicable requirement is assigned to modules that implement functionality referenced by the requirement.
- For further break down of the assignment, individual test objects are linked to the requirements. This makes sense if the module has a large number of linked requirements.
- At last there is a small subset of all available requirements that must be verified. The requirement linking must be further broken down to test case level for a given test object.

For this process TESSY provides the Requirement Coverage view within the Overview perspective.

It is divided into two tabs:

- The Planning tab is the editor for all requirement links to modules, test objects and test cases.
- The Execution tab provides quick overview about the achieved test results for linked requirements.

The content that is displayed depends on the current selection. You can select the following elements to display data:

selected element	**content displayed**
document, folder	Displays the requirement links of all sub elements and provides an overview of which other sub elements have not been linked to those requirements.
module, test object	If not already linked with any requirement, all available requirements will be displayed; otherwise only the linked requirements will be displayed. You can choose to display all available requirements by clicking on "Always show unlinked requirements" button (this button remains active for other selections as well).
test case	Displays only the requirements for the currently selected test case (and those requirements already linked to the module or test object). You can choose to link the module and test object level linked requirements also on test case level, if applicable.

Table 6.86: Selected elements within Requirement Management and displayed contents

Important: When selecting objects on upper levels of the test project, the calculation of the test planning/execution links can take a moment.

Planning tab

The current status of the links between modules, test objects, test cases and requirements reflects the current state of your requirements coverage. This coverage can be examined

on arbitrary levels of your test project.

You can also create a report that shows the currently achieved planning coverage.

Execution tab

After execution of any tests, the test results are stored within test runs. The test result of a test run covers the requirements that were linked to modules, test objects or test cases at the time the test was executed. Therefore, the actual execution coverage result may differ from the planning coverage result. The execution coverage view is read-only, because this just displays the results. Any changes to requirement links need to be carried out within the planning coverage view.
You can create a report that shows the currently achieved execution coverage.

6.4 TEE: Configurating the test environment

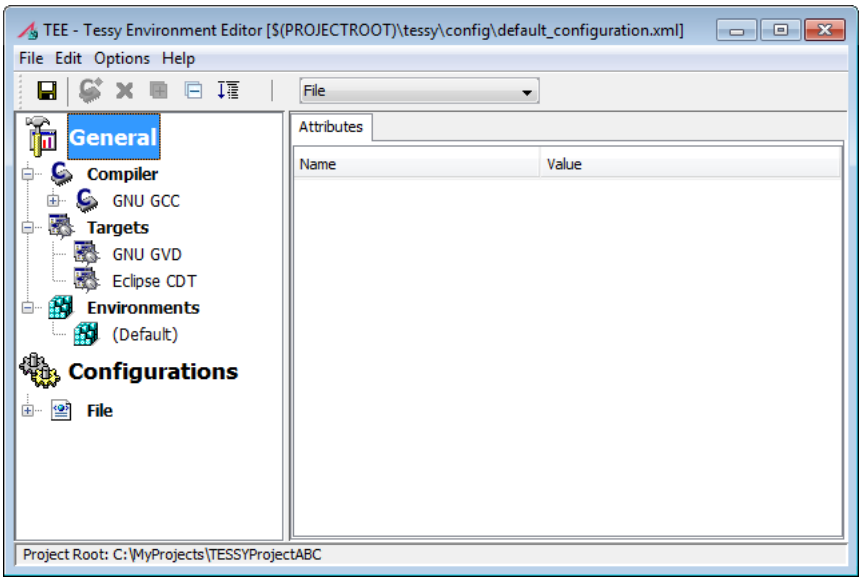


Figure 6.61: Test Environment Editor (TEE)

The test environment of a module specifies the common settings to execute a test run, such as:

- target compiler and debugger combinations
- additional compiler and linker options
- source files

We do that within the "Test Environment Editor", the TEE.

With the installation of TESSY, the configurations for all supported compiler and target environments (including necessary settings and files) were copied to the TESSY installation directory. You need to enable the compiler and targets that you want to use. Their default settings may need to be adapted to your needs, i.e. the installation path of the compiler or target debugger is one of the settings that normally need to be changed to your local values. Settings which have already been used with a previous version of TESSY were also taken over during installation.

The TEE configuration management allows you to create variants of compiler and target settings and assign them to a module. You can also save your settings in a specific configuration file to use this file with a given project database. This allows easy sharing of specific environment configurations between developers of the same development team.

As a result you have all your basic settings at one central place, e.g. include paths, additional compiler options, etc. Once configured, you can start testing immediately using the required configuration for all your modules.

6.4.1 Starting the TEE

To open the TEE:

→ In the menu bar click on "File" > "Edit Environment".

The TEE will start either with a specific custom configuration file (see section 3.1.2 Creating a configuration file) or with a default configuration provided by the TEE automatically.

6.4.2 Structure and handling of the TEE

Important: TEE is not an eclipse-based program and does not handle with views and perspectives! Therefore you cannot drag and drop the panes.

pane	**location** (default)	**function**
General and Configurations	left	Paragraph General lists all supported compiler and target environments which were configured with default settings. Paragraph Configurations lists all supported compiler and target combinations. The settings of these combinations have been inherited by the paragraph General.
Attributes	right	To display all attributes of an item as soon as it was selected in the left pane.

Table 6.89: Structure of TEE

Per default, only basic attributes are visible for the user. You can display the settings in various ways, e.g. showing all attributes or hide all disabled or inactive settings:

→ In the tool bar select an option fron the pull-down menu (see figure 6.62).

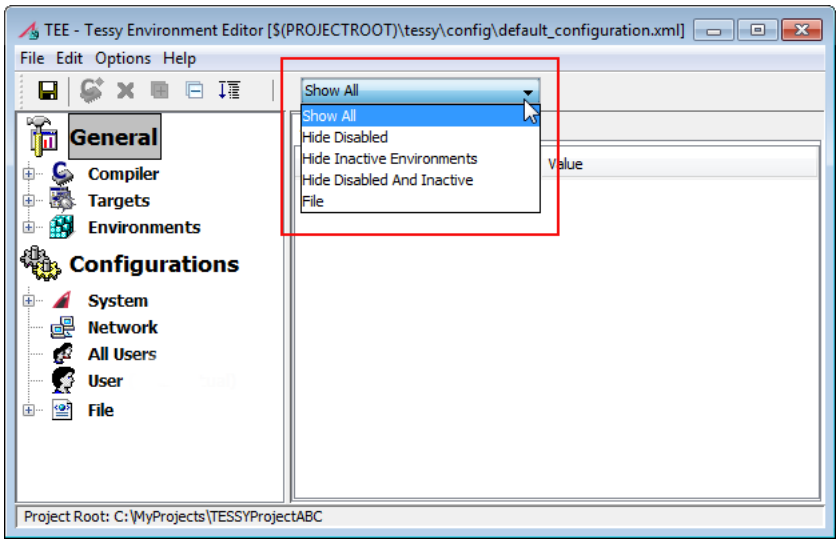


Figure 6.62: Modes of TEE to display the settings

When you select any item on the left pane, the attributes will be shown in the right pane (see figure 6.63).

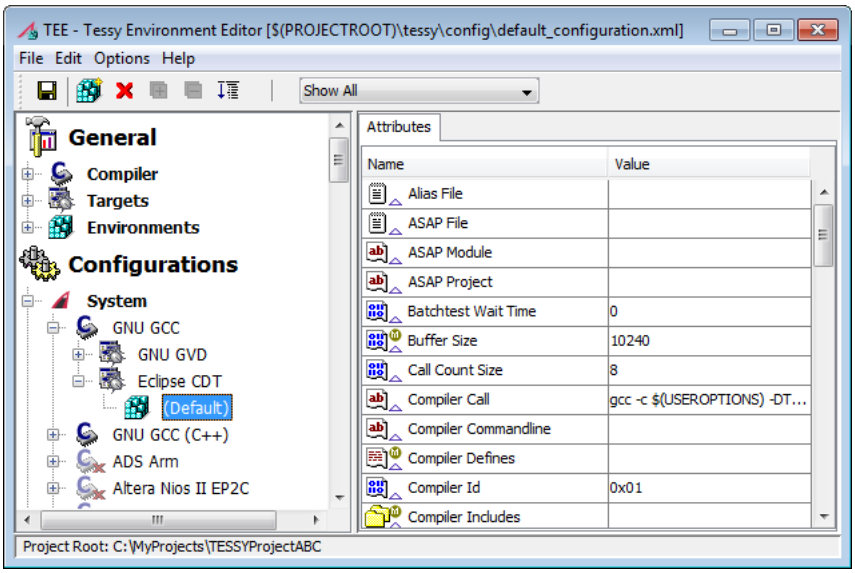


Figure 6.63: Environments of the Eclipse target

Tool bar handling

The tool bar preserves various handlings, depending on the selected item:

To expand or retract,

→ select an item and click on (Expand) or (Retract).

Adding compiler, target, environment

To add a compiler, target or environment, which is only possible in the Configurations pane,

→ select a compiler, a target or an environment.

The case sensitive icon in the tool bar will change to either (Adding Compiler), (Adding Target) or (Adding Environment).

→ Click on the icon. The element will be added.

Deleting an item

To delete any item,

→ select the item and click on (Delete).

You can refresh the panes in TEE by pressing F5.

6.4.3 Status indicators

The following table shows the indicators of status and their meaning which are used by TEE.

indicator	status / meaning
	item added to execution environment
	item added to path variable of the execution environment
	item disabled
	item disabled and another item overwrites (or replaces) it within the configuration hierarchy

continue next page

indicator	status / meaning
	error
	error, but the cause is somewhere within the attributes inherited by item. Check for errors upwards or downwards within the configuration hierarchy and fix the error there.
	information
	the information comes from another item upwards or downwards within the configuration hierarchy
	attribute inherited (The attributes in paragraph Configuration, e.g. compiler, targets, environment, are normally inherited from the respective paragraph General.)
	compiler active
	compiler inactive (probably overwritten or replaced by another item downwards the configuration hierarchy)
	target active
	target inactive (probably overwritten or replaced by another item downwards the configuration hierarchy)
	environment active
	environment inactive (probably overwritten or replaced by another item downwards the configuration hierarchy)
	warning of error within the attributes of the item (need to be resolved somewhere else within the configuration hierarchy)
	warning, but the cause is somewhere within the attributes inherited by this item. Check for warnings and errors upwards within the configuration hierarchy and fix the errors there.

Table 6.94: Status indicators of TEE

Example:

	The directory is added to path variable and to execution environment. The attribute has been inherited.

Different fonts as indicators

TEE will display the attributes in different fonts to indicate the following situations:

normal letters	represent factory settings respectively default settings from paragraph "General" and have been inherited.
bold	value has been defined the first time for the attribute
italic	factory setting respectively default setting from paragraph "General" has been overwritten

6.4.4 Configuration files

All displayed settings will be stored in a set of configuration files with the extension *.xml. Every configuration group (System, Network, etc.) of the paragraph Configuration has separate configuration files.

> The configuration file of System contains the default settings for all supported compiler and target environments and has been installed with TESSY. As soon as the TEE is launched, a configuration file will be generated for the user. The configuration files for User respectively All Users will be empty, as long as no configurations were created, e.g. copied into this subsection.
> The configuration file for User / All Users contains only compiler and target configurations which you have copied into these subparagraphs.
> The TEE will always merge the configuration files during startup to display the resulting settings for the respective configuration groups.

The configuration files will be stored in following default folders:

paragraph	**storage, file(s), function**
System	stored under `C:\Program Files\Razorcat\TESSY\config` • `configuration.xml`: Contains settings for the "General" and "Configurations" paragraphs. • `configuration.default.xml`: Presents the factory settings of TESSY. During installation of new TESSY versions, the settings within this file will be used to update configuration.xml if necessary.

continue next page

paragraph	storage, file(s), function

Network	stored under `C:\Program Files\Razorcat\TESSY\config`
	• `network_configuration.xml`: Specifies a configuration.xml which was stored in your network. The syntax is as follows:
	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<link target="[PATH]\\configuration.xml"/>`
	We recommend to use a custom configuration file instead of a configuration file in your network. This is still backward compatible with former versions of TESSY.

All Users	stored under `%ALLUSERSPROFILE%\Application Data\Razorcat\TESSY\[Version]\config` • `configuration.xml`: Contains settings for this section (if any).
User	stored under `%APPDATA%\Razorcat\TESSY\[Version]\config\` • `configuration.xml`: Contains settings for this section (if any). • `configuration.system.xml`: Contains all your changes made in the "General" and "Configurations" paragraphs if you do not have enough privileges (restricted user) to change the system configuration file of the TESSY installation. Another user account on the same computer will not be updated.

Table 6.97: Contents, functions and storage location of configuration files

6.4.5 Using default configurations

The TEE provides predefined configurations for all supported compiler and target environments:

→ Click on "Configuration" > "System".

The settings of these combinations are inherited from the paragraph General.

By default, at least the GNU GCC and GNU GCC (C++) environments will be enabled (and all your compiler/target environments used in previously installed versions of TESSY, if any). All other predefined configurations are disabled (see figure 6.64).

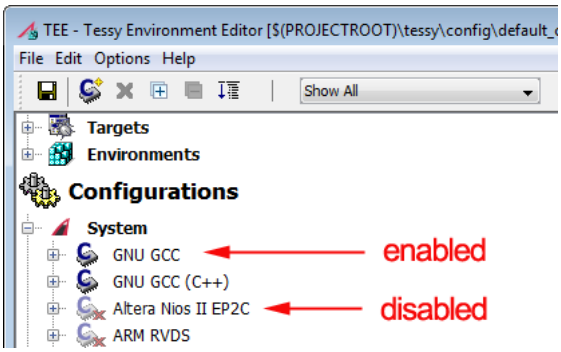


Figure 6.64: Enabled and disabled configurations within TEE

6.4.6 Enable configurations

Enable configurations

With exception of the used source files and the GNU toolset, you have to enable and adjust common settings of a specific test environment. After an appropriate compiler/target combination has been enabled, it will be available within the Properties view of the Overview perspective (see section 6.1.3 Properties view).

A configuration consists of a combination of compiler, target and a default environment.

You have to options:

- **Either** you enable both a compiler and a target in paragraph General.
- **Or** you enable directly the desired configuration within the paragraph System.

Enable compiler and target in paragraph General

To enable the compiler,

→ select "Show All" from the pull-down menu to display all supported compiler and target environments.

→ In paragraph General open the compiler tree and select the compiler.

→ Select "Enable Compiler" from the context menu or press Ctrl+E (see figure 6.65).

TEE will remove the red circle with the white cross from the icon to indicate, that this compiler is enabled.

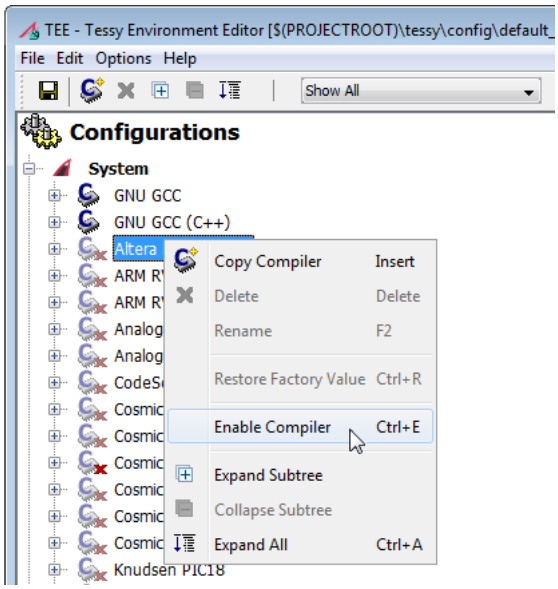


Figure 6.65: Enable compiler

To enable target,

→ In paragraph General open the target tree and select the target.

→ Select "Enable Target" from the context menu or press Ctrl+E.

TEE will remove the red circle with the white cross from the icon to indicate, that this target is enabled.

Enable configuration within paragraph System

You can only use supported compiler/target combinations.

The system tree contains all supported compiler/target combinations, e.g.
(Compiler) Tasking XC166
(Debugger) Tasking Crossview
(Environment) (Default)

→ Right click the (default) environment.

→ Select "Enable Environment".

6.4.7 Customizing enabled configurations

Normally you need to change some settings for your specific environment. Some of the settings will be checked for validity. The TEE will check all enabled configurations and displays error and warning signs as soon as an error has been found, i.e. if the "Compiler Install Path" must be corrected.

Adding and editing attributes

TEE preserves all default settings. You can revert the default values with "Restore Factory Value" in the context menu (right click the attribute).

Editing attributes

To edit an attribute,

→ right click an attribute in the Attributes pane.

→ Select "Edit Attribute" from the context menu.

→ If you want to change only an attribute value, select "Edit Attribute Value" from the context menu.

Depending on the attribute type, either a standard selection dialog for that kind of information will appear (e.g. Browse for Folder in case of directories) or the inline editor for the value will be activated.
If you have changed a default value other than the factory setting, the attribute will be displayed in italics.

→ If you want to delete the attribute value, select "Reset Attribute" from the context menu or press Del.

This will either remove the local value and show the inherited value or delete the whole attribute entry, if it is only defined locally in this section.

Adding attributes

To add an attribute,

→ right click in the Attributes pane.

→ Select "New Attribute" from the context menu.

The New Property dialog will open.

You see different attribute types available: String, Boolean, Number, Float, File and Directory.

→ Specify the desired type.

Important: The type cannot be changed once the attribute is created!

→ Check the desired specific attribute flags. This depends on the type used. For description see table below.

→ Click "OK".

flag	**description**
Inheritable	This flag will always be ticked by default. It controls the inheritance of the attribute: The attribute will be available in all (child) section nodes. Some basic attributes are defined at the main nodes, e.g. Compiler. Each supported compiler will inherit these basic attributes.
validate	This flag may be important for Directory or File types. The attribute value will be validated by TEE (e.g. whether the path respectively the file is available). An error sign will indicate that the file or directory could not be found.
Read only	This flag makes it impossible to change a default value by using the attribute pane of the module properties dialog.
Always Valid	Internal flag of TEE
As List	Using this flag, the attribute value will be handled as list of values (comma separated). The values may be edited using a special list dialog. This is useful for File or Directory types.
Multiline	Provides a text window for Multiline editing.
System	Internal flag of TEE.
Hex Format	This flag is useful in combination with the Number type. TEE will convert all inputs (i.e. a decimal value) to a hex value, e.g. 1 > 0x01.
Visible	This flag makes the attribute visible in the attribute pane of the module properties dialog (and within the test report).
Not Empty	Checks whether the value is not empty. An error sign will indicate that the attribute does not have a value.
System	Internal flag of TEE.

continue next page

flag	description
Environment Variable	This flag is useful during test execution and during the make process: TESSY will create an environment variable within the process space of the process that will be used for test execution (e.g. running the slave process) and for make (e.g. building the test driver). **Important:** The environment variable will only be recognized by TESSY if a plus-sign is used in front of the Make Call value.
Add to PATH Variable	This flag is useful for attributes of type directory. Like described above for the flag "Environment Variable", the respective directory value will be added to the PATH variable of the process space used for test execution and make.
Makefile Variable	Adds this variable to the generated makefile for compilation/linking of the test driver application. You can use this variable within the makefile for include paths or other settings required during the make process. A variable named "My Include Path" will be added to the generated makefile as MY_INCLUDE_PATH with the respective value.

Table 6.105: Means of flags in the attribute properties

6.5 TIE: Preparing the test interface

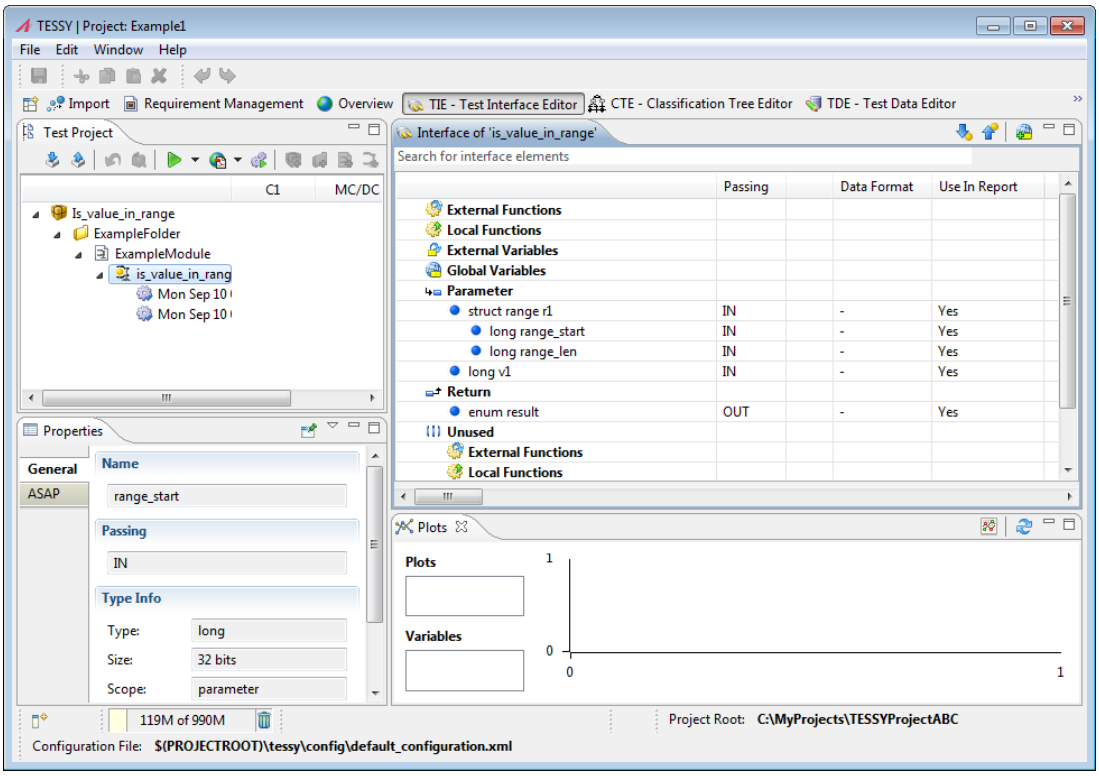


Figure 6.66: Perspective TIE - Test Interface Editor

After configuring the test environment of a module you can open the module which starts the analysis of the respective source files. The functions found within the source files will be available as test objects and need to review and maybe update the test object interface settings.

Within the TIE you determine which values are input and which ones are output variables. **Input values** are all interface elements that have to be set before execution of a test object. **Output values** are compared with the expected values after test execution.

You might need to specify missing information that TESSY could not determine automatically, i.e. array dimensions or values of enumeration constants. This can happen due to the usage of the "sizeof" operator when declaring arrays or enumeration constants. Upon opening the module, TESSY will try to assign useful default passing directions automatically. You can change these default interface settings to your needs.

6.5.1 Structure of the TIE perspective

pane	**location** (default)	**function**
Test Project view	upper left	Same view as within the Overview perspective.
Properties view	lower left	Same view as within the Overview perspective.
Interface view	upper right	To display all interface elements of the test object and to provide the edit fields to enter passing directions of variables as well as additional information.
Plots view	lower lower	Same view as within the TDE perspective.

Table 6.106: Structure of TIE

6.5.2 Test Project view

6.1.2 Test Project view

The Test Project view displays your test project which you organized within the Overview perspective.

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

6.5.3 Properties view

Passing directions of a variable

The Properties view is context sensitive: You can view the passing direction of a variable (e.g. IN, OUT, IRRELEVANT) if you select the variable within the Interface view. Then the Properties view will display the passing direction and the type information (see figure 6.67).

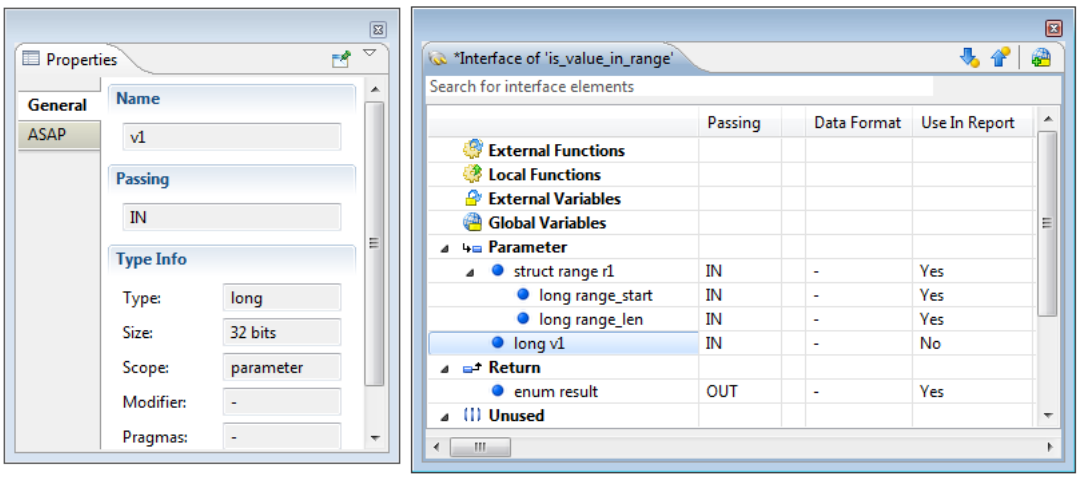


Figure 6.67: Information of passing direction and type

6.5.4 Interface view

Handling

You can browse through the interface items of the currently selected test object. An arrow in front indicates further levels (see figure 6.68).

Important: In some windows versions you need to move the mouse over the view to see the arrows! They will fade-out when the view is not active.

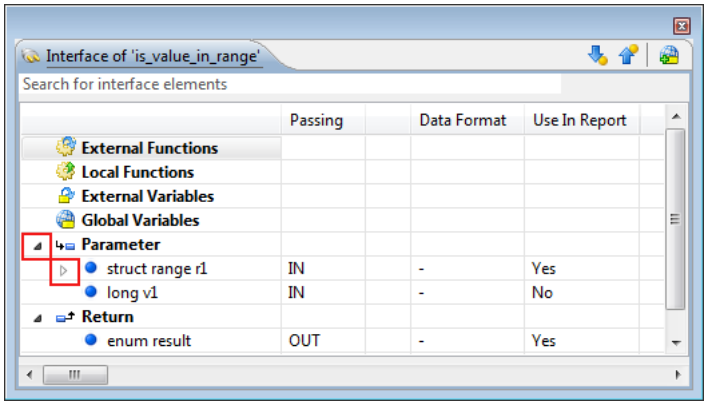


Figure 6.68: White arrow indicating further levels, black arrow when expanded

Icons of the view tool bar

icon	action / shortcut / comment
	Highlights the next undefined value.
	Highlights the previous undefined value.
	Creates a new variable.

Table 6.107: Tool bar icons of the Interface view

Interface elements

Interface elements

The variables are either read within the function (IN), written within the function (OUT), both read and written (INOUT), to be altered by usercode (EXTERN), or they are simply not used within the function (IRRELEVANT).

The TIE classifies all recognized interface elements of the test object into the following sections:

External Functions	All called functions of the test object. These functions are not defined within the source file(s) of the module.
Local Functions	All functions defined within the source file(s) and called from the test object.
External Variables Global Variables	These sections list the external and the global variables. The external variables are all used variables, which are not defined within the source files of the module.
Parameter	Parameter of the test object. You can browse through the structures by clicking on the plus sign to see the basic components.
Return	Return value of the test object. In our case defines an enum variable values for “yes” or “no”.
Unused	These sections will be explained in more detail later in this manual.

Setting passing directions

Setting passing directions

→ Click in the relevant cell of the element in the column "Passing".

A drop-down menu will be displayed with the available options IN, OUT, INOUT and IRRELEVANT.

→ Select a suitable passing direction.

The passing direction reflects the kind of usage for each variable while testing the test object. You can specify how TESSY treats a value for an interface variable either to provide the value before test execution (IN) or to keep the value for evaluation and reporting after test execution (OUT).

You have to specify one of the following passing directions for each interface element:

- provide an input value for that interface element, because the element is only read by the test object (IN),
- evaluate and report the results of that interface element, because the element is only written by the test object (OUT),
- both provide a value and evaluate the result, because the interface element is both read and written by the test object (INOUT),
- provide a value within the UCE (Usercode Editor) of TESSY (EXTERN) . With this setting, the interface element is visible in the scope of the user code and may be set using C code,
- not use the interface element at all (IRRELEVANT). In this case, you will not see this variable for all further testing activities.

The following table shows possible passing directions of the different types of interface elements:

direction: / **element:**	**IN**	**OUT**	**INOUT**	**EXTERN**	**IRRELEVANT**
external variable	x	x	x	x	x
global variable	x	x	x	x	x

continue next page

direction: / element:	IN	OUT	INOUT	EXTERN	IRRELEVANT
parameter	x			x	x
return		x		x	x

Table 6.112: Possible passing directions of the interface elements

Automatic analysis of the passing directions

During processing when opening the module, TESSY analyzes the passing directions automatically and stores its findings in the interface database. This information is available in the TIE as default values of the passing directions. TESSY analyzes the usage of individual interface elements by the test object.

Warning: Although TESSY usually correctly recognizes all interface settings, open the TIE for every test object and make sure that the values are set correctly or do match your needs!

Depending on that usage, the following passing directions will be set as default:

read only	IN
write only	OUT
read and write	INOUT
not used	IRRELEVANT

In case that the passing directions or any other interface information could not be determined the respective fields in the TIE will be empty, i.e. if TESSY could not calculate the size of an array dimension (indicated with a question mark), you have to set them manually. **Reset passing direction to the initial value** TESSY analyzes the usage of individual interface elements by the test object. Change the passing direction of an interface element to suite your needs.

Reset the passing direction for all interface elements of one section:

→ Select the respective section and "Reset to Default Passing" from the context menu (see figure 6.69.

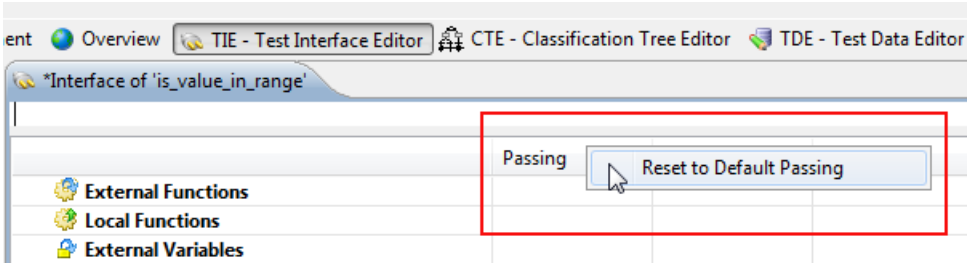


Figure 6.69: Resetting passing directions

Reset the passing direction only for an individual interface element:

→ Select the respective interface element and "Reset to Default Passing" from the context menu.

Setting the data format

Setting the data format

You can change the data format:

→ In the row "Data Format" click into the cell to open the pull-down menu.

→ Click on a format to change it either to "Decimal", "Hexadecimal" or "Binary" (see figure 6.70) and click on (Save).

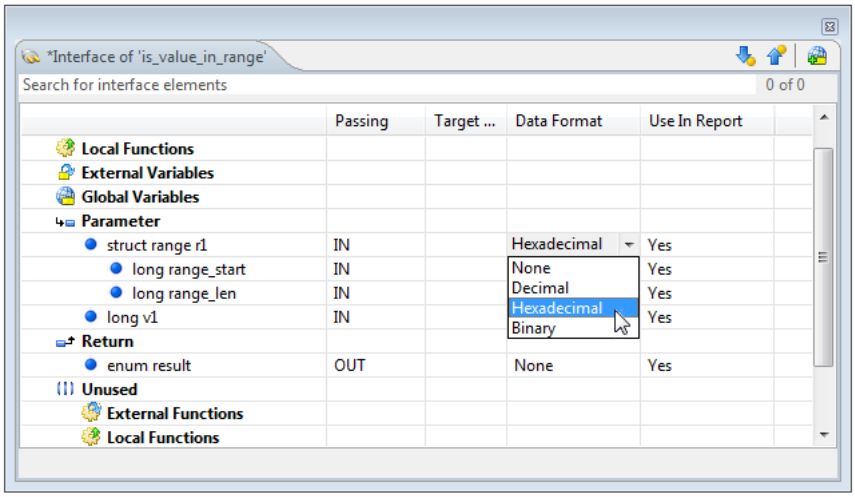


Figure 6.70: Setting the data format

Important: If you change the data format, all newly entered values within the Test Data view of the TDE will be formatted into the new format. Existing data will not be formatted!

Setting passing direction of special data types

Pointers and complex data types will be treated slightly different as normal data types.

Pointers

Interface elements of pointer type have two passing directions:

Pointers

Both the passing direction of the pointer itself and the passing direction of the target to which it points have to be specified. The passing direction of the pointer and the target can be set interdependently, but they are checked or corrected by TIE to ensure valid combinations.

Complex data types

Complex data types as "Structure" and "Union" have a dependency between their passing direction of the overall structure/union and the passing directions of their components. To avoid invalid combinations the TIE checks the setting of passing directions for these data types in the following manner:

- When the passing direction of one component is set, TIE determines the resulting passing direction for the overall structure/union and set them automatically.
- When the passing direction for the overall structure is set, all components are automatically set to the same passing direction.

Arrays

The passing direction of the data type "Array" will be set for the entire array to the same direction. Only one passing direction will be defined for the whole array and all elements.

Arrays

If the array is made up of structured array elements (e.g. structures) it is possible to define different passing directions for the components of these structures.

Defining stubs for functions

The TIE displays all functions used by the test object either in section External Functions or Local Functions and it provides an interface to define stubs for these functions that will

be executed instead of the original function. TIE distinct two different stub functions:

- A stub function for which you can enter C code.
- A so called advanced stub function that allows to provide values for parameters and return values of stub functions like normal variables in the TDE.

You can define stubs globally for all test objects of the module or create a stub independently of the global module setting.

Creating a stub function

To create a stub,

→ rightclick the function and choose "Create Stub" from the context menu (see figure 6.71).

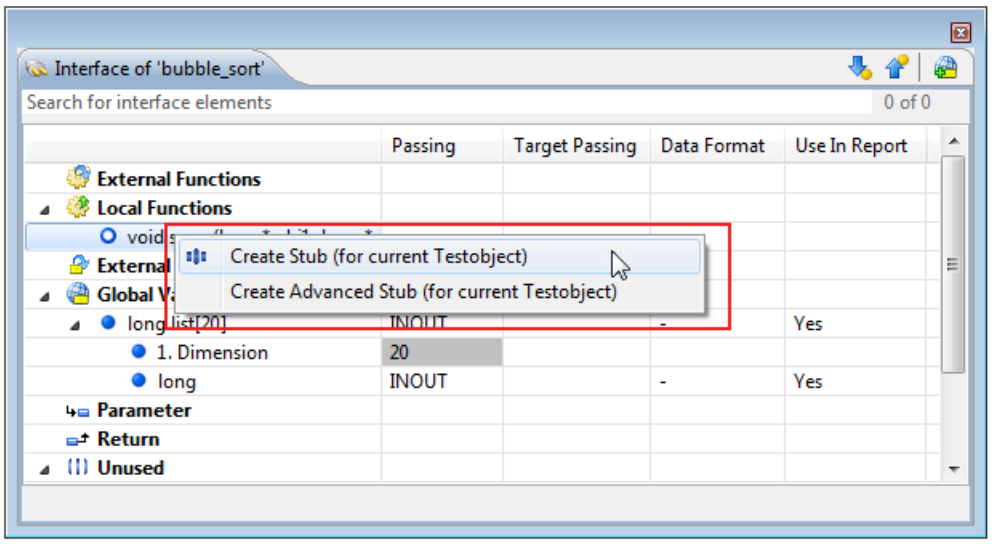


Figure 6.71: Create a stub function within the context menu

You can create stubs either for external or local functions which will be executed instead of the original functions. There are several options available:

- Create stubs for all functions at once for all test objects of the module (global setting).
- Create stubs for a single function for all test objects of the module (global setting).

- Create stubs for the current test object.
- Change settings for the current test object.

The enhancement to normal stub functions are advanced stub functions, that allow to provide values for parameters and return values of stub functions like normal variables. TESSY checks if the stub is called by the test object with the specified parameter values, otherwise the test fails. You can also provide a return value for further processing by the test object. This reveals if the test object handles the return values of the stub function correctly. To create an advanced stub,

→ rightclick the function and choose "Create Advanced Stub" from the context menu.

Creating an advanced stub function

You can create advanced stubs either for external or local functions. There are several options available:

- Create advanced stubs for all functions at once for all test objects of the module (global setting).
- Create advanced stubs for the current test object.
- Change settings for the current test object.

Other interface settings

For test execution the information on data types of the test object interface has to be complete. The dimensions for arrays, the values of the enumeration constants for enumerations, and the dimensions for bitfields have to be defined. If these values have been automatically recognized by TESSY while opening the module, the respective text field will show the calculated value for every data type. In this case, it is not possible to change these values.

If a value for an interface element has not been recognized automatically, the respective text field will be empty or contain the value -1. In case of arrays TIE will also use question marks to indicate this issue, i.e. array[?]. In all those cases you have to add values manually.

Warning: Wrong array dimensions or wrong values for enumeration constants can cause the test object to crash during test execution! TIE cannot check for plausibility of used values!

6.5.5 Plots view

The Plots view visualizes graphically the test results. For details refer to section 6.7.11 Plots view within chapter TDE: Entering test data.

6.7.11 Plots view

6.6 CTE: Designing the test cases

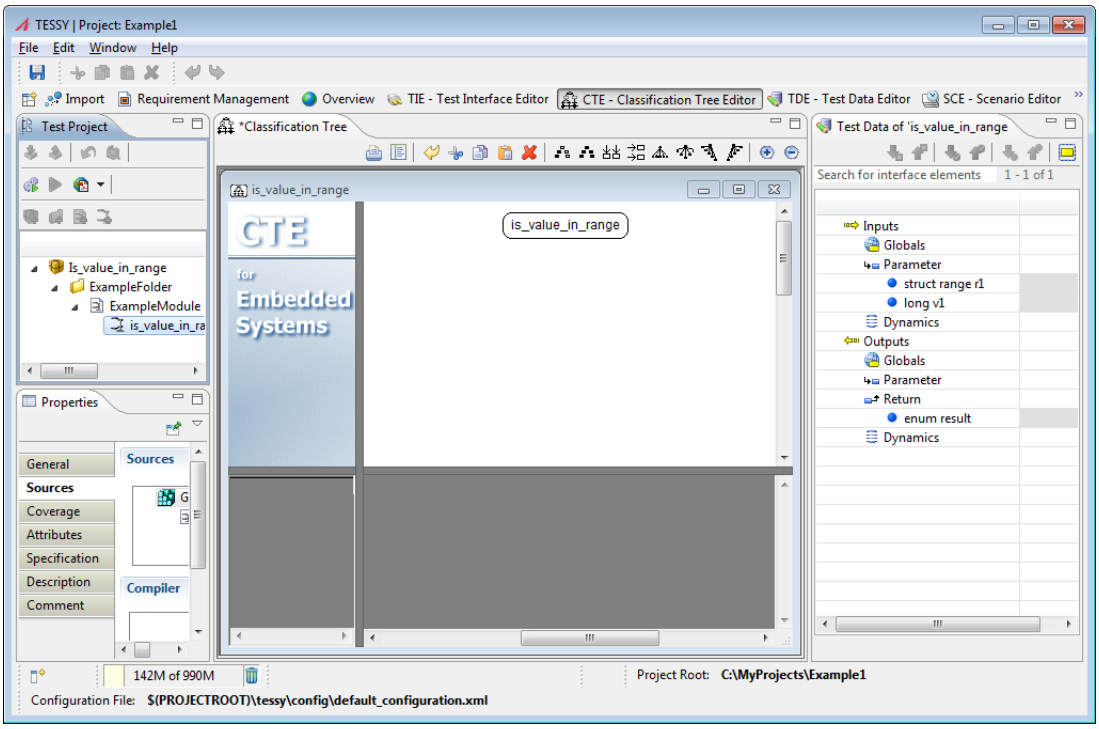


Figure 6.72: CTE perspective

6.6.1 The basic idea

After preparing a test in the TIE, you need to create well designed test case specifications. The Classification Tree Method provides a systematical approach to create test case definitions based on the functional specification of the function or system to be tested. TESSY includes the specialized Classification Tree Editor CTE which assists you in creating low redundant and error sensitive test cases.

The basic concept of the Classification Tree Method is to first partition the set of possible inputs for the test object separately and from different aspects, and then to combine them to obtain redundancy-free test cases covering the complete input domain.

6.6.2 Starting CTE

→ Switch to the CTE perspective.

6.6.3 Structure of the CTE perspective

pane	**location** (default)	**function**
Test Project view	upper left	Displays your test project. For editing your test project switch to the Overview perspective.
Properties view	lower left	Displays the properties of your test project, e.g. sources to the test object.
Classification Tree view	center	To edit the classification tree.
Test Data view	right	Allows to assign test data to classification tree elements.

Table 6.124: Structure of the CTE perspective

6.6.4 Test Project view

The Test Project view displays your test project which you organized within the Overview perspective.

6.1.2 Test Project view

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

6.6.5 Properties view

The Properties view displays all the properties which you organized within the Overview perspective. Most operations are possible.

6.1.3 Properties view

For changing a source switch to the Properties view within the Overview perspective.

6.6.6 Classification Tree view

Classification Tree view

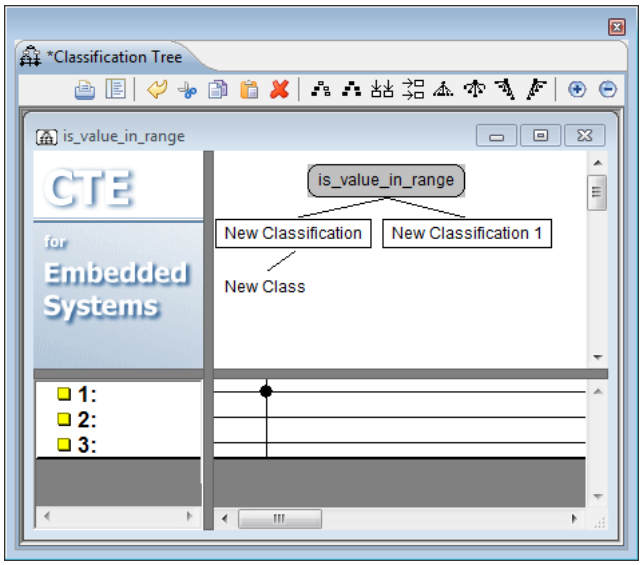


Figure 6.73: Classification Tree view

Icons of the view tool bar

icon	action / shortcut / comment
	Prints the classification tree (Ctrl + P).
	Gives a print preview.
	Undoes the last move or edit operation within the classification tree pane.
	Cuts the selection (Ctrl + X).
	Copies the selection (Ctrl + C).
	Paste (Ctrl + V).
	Delete (Del).
	Selects all leaves that are children of the current selection (Ctrl + L).
	Selects the whole CT (Ctrl + A).

continue next page

icon	action / shortcut / comment
	Adjusts the tree horizontal (Shift + H).
	Adjusts the tree vertical (Shift + V).
	Displays the tree in a horizontal layout.
	Displays the tree in an arrow layout.
	Displays the tree in a leftdown layout.
	Displays the tree in a rightdown layout.
	Zooms in.
	Zooms out.

Table 6.125: Tool bar icons of the Classification Tree view

Structure of the Classification Tree view

pane	**location** (default)	**function**
test item list	lower left	Defining test cases, test sequences and test steps. Every test item creates a new line in the table pad.
draw pad	upper right	Drawing the classification tree with a root, classifications and classes.
table pad	lower right	Marking classes of the classification tree in order to define test cases, test sequences and test steps. Every test item creates a new line in the table pad.

Table 6.126: Structure of Classification Tree view

Designing a classification tree

A test case is formed through the combination of classes from different classifications. For each test case exactly one class of each classification is considered. The combined classes must be logical compatible; otherwise the test case is not executable. You should

choose as many test cases that all aspects, not only individual but also in combination, are considered enough.

The method offers a graphical representation of the recursive partitioning of classifications and classes as classification tree. The classifications are drawn as named rectangles. The respective classes are arranged below. To specify the test cases as combination of classes the classification tree is used as the head of a combination table wherein the classes, which are to be combined, are marked.

Creating classifications and classes

To create classifications:

→ Select the root tree item.

→ Use the context menu ("New" > "Classification", see figure 6.74) or press Ins.

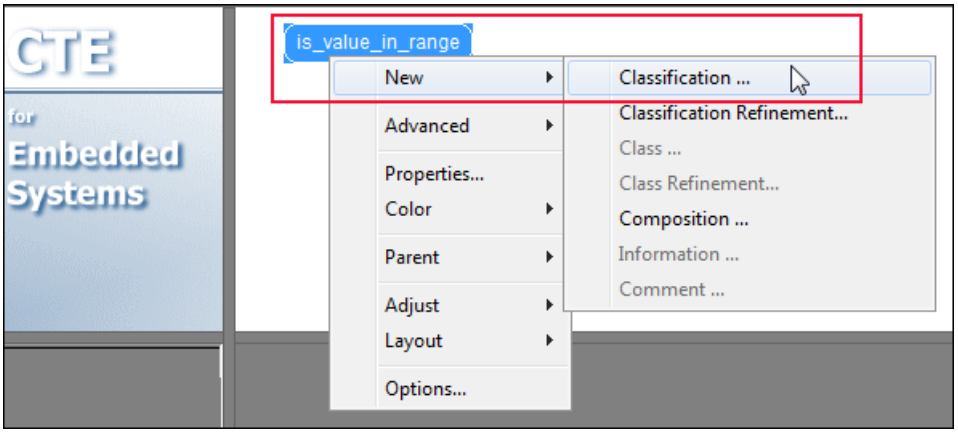


Figure 6.74: Creating a new classification with the context menu

To edit the classification:

→ Double click the new classification or press F2 to start the inline editor for the tree item.

Within the draw pad you can move the classifications with drag and drop: Select either a classification, a sub tree (click on) or select all (click on) and press the mouse button until the cursor turns into a cross with four arrows , then move the selection.

To create classes:

→ Right click the classification and select "New" > "Class" or select the classification and press Ins.

Try the several layouts for the tree vor a better overview, e.g. (Arrow Layout), leftdown or horizontal !

Notice the following habits:

- All tree items with assigned test data are marked with a yellow dot, when not selected.
- When selecting a tree item, you will see the test data entered for this item within the Test Data view.
- When selecting any interface element within the Test Data view, all classification tree elements that contain test data for this interface element will be marked with a blue dot.

Creating test cases

To create test cases:

→ Select the pane on the lower left (test item list).

→ Create the test cases either using the context menu ("New" > "Classification", see figure 6.75) or press Ins.

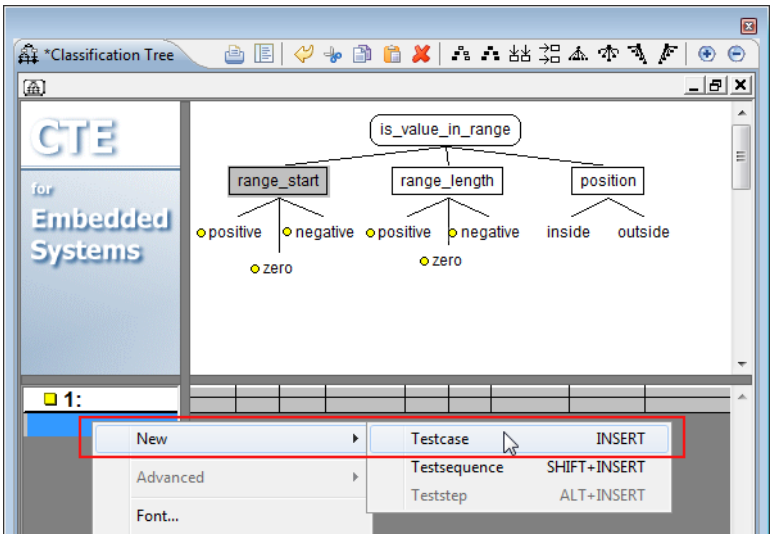


Figure 6.75: Creating test cases in the test item list

Test cases are defined by setting markings in the combination table:

→ Move the mouse pointer over the combination table.

Connecting lines to the classes of the classification tree are drawn. If the mouse pointer is placed over a point of intersection it is changed to a circle.

→ Click on a circle to connect a test case with a class.

Important: If you connect a test case with a class, the test data for the test case is displayed within the Test Data view. The test data is read-only because it is defined by the marks set within the combination table.

→ Click on to save the classification tree.

→ Switch to the TDE perspective.

Please notice the following habits:

- Test items with values stemming from the CTE perspective are marked with new status indicators: (test case) and (test steps).
- Values stemming from the CTE are read-only. If you want to change them, switch back to the CTE perspective and do your changes there.

6.6.7 Test Data view

Whether using the CTE or creating the test cases manually within the TDE perspective, you will use the Test Data view to enter the values. Because some operations and overviews are only possible within the TDE perspective, switch to chapter 6.7.7 Test Data view to learn how to use the Test Data view.

6.7.7 Test Data view

6.7 TDE: Entering test data

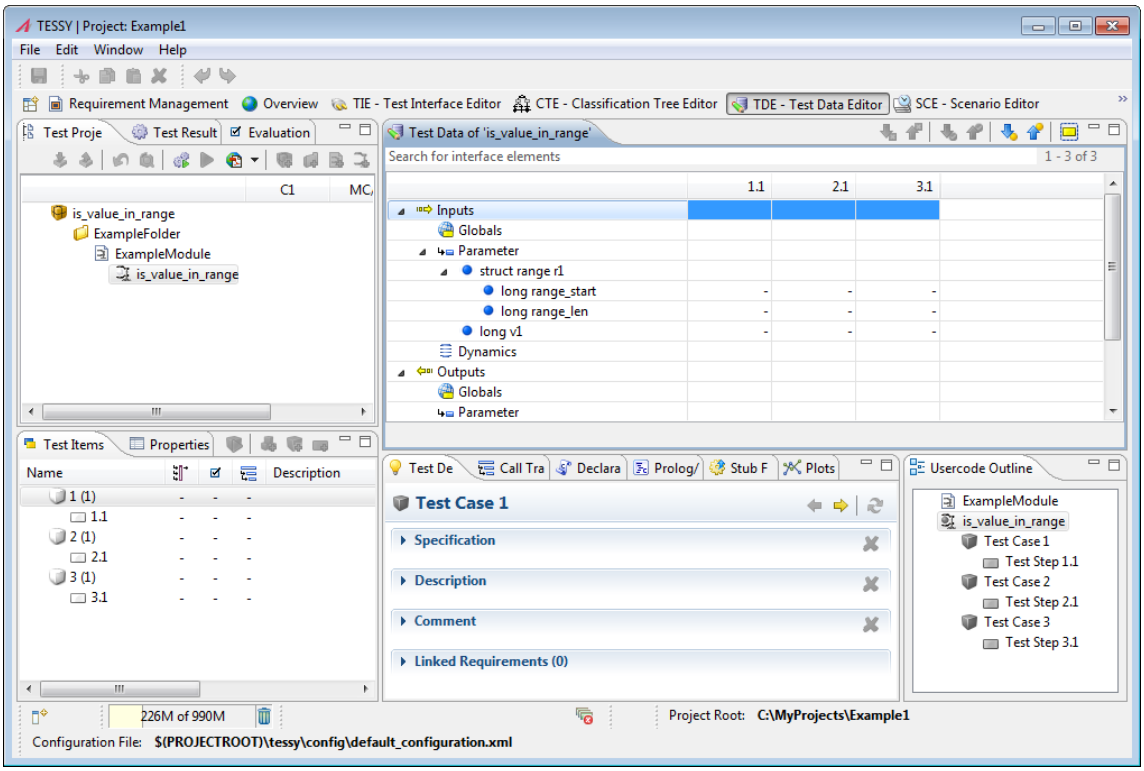


Figure 6.76: TDE perspective

6.7.1 Structure of the TDE perspective

pane	**location** (default)	**function**
Test Project view	upper left	Same view as within the Overview perspective.
Test Results view	upper left	Same view as within the Overview perspective.
Evaluation Macros view	upper left	Same view as within the Overview perspective.
Test Items view	lower left	Same view as within the Overview perspective.
Properties view	lower left	Same view as within the Overview perspective.

continue next page

pane	**location** (default)	**function**
Test Data view	upper right	To enter test data and expected values, after the test execution, reviewing passed, failed and undefined values.
Prolog/Epilog view	lower center	To specify usercode that will be executed at a certain point in time during the test execution.
Usercode Outline view	lower right	To display the usercode that will be executed at a certain point in time during the test execution.
Plots view	lower center	To visualize graphically the test results.

Table 6.137: Structure of TDE

Usercode

TESSY provides an interface to specify the usercode that will be executed at a certain point in time during the test execution. Using the usercode views (i.e. Prolog/Epilog view and Usercode Outline view) you can specify such C code fragments or emulator scripts depending on the selected target configuration.

6.7.2 Test Project view

The Test Project view displays your test project which you organized within the Overview perspective.

6.1.2 Test Project view

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

6.7.3 Test Results view

The Test Results view displays the coverage measurement results and the results of a test run of expected outputs, evaluation macros and call traces, if applicable. It is the same view as within the Overview perspective.

6.1.6 Test Results view

6.7.4 Evaluation Macros view

6.1.7 Evaluation Macros view

This view lists the detailed results of the evaluation macros if the usercode of the test object contains any evaluation macros. The results are displayed wherever they occur within the usercode, e.g. within stub functions or test step epilogs. It is the same view as within the Overview perspective.

6.7.5 Test Items view

6.1.5 Test Items view

Within the Test Items view you get an overview about your test cases and test steps which you organized within the Overview perspective or the CTE (see section 6.6).

To create test cases and test steps manually without using the Classification Tree Editor, switch to the Test Items view within the Overview perspective.

6.7.6 Properties view

6.1.3 Properties view

The Properties view displays all the properties which you organized within the Overview perspective. Most operations are possible.
For changing a source switch to the Properties view within the Overview perspective.

Type information of a variable

The view is context sensitive: You can view the passing direction and all type information of the variable (i.e. the basic type, the size as well as any modifiers and pragmas) if you select the variable within the Test Data view (see figure 6.77).

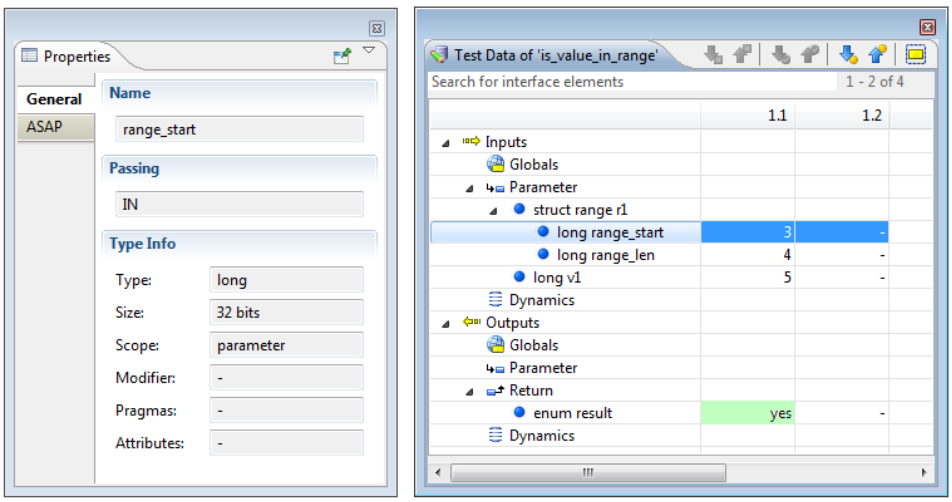


Figure 6.77: Type information of the variable long range_start

6.7.7 Test Data view

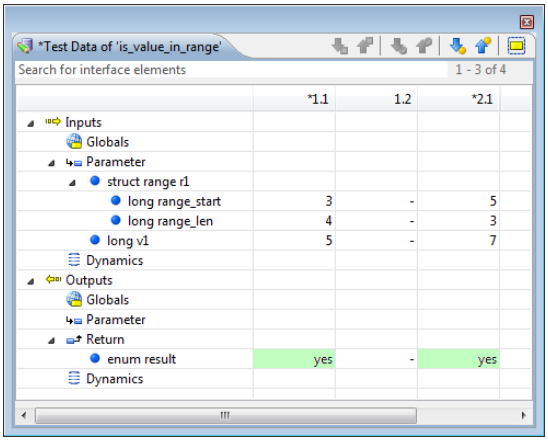


Figure 6.78: Test Data view

Whether using the CTE or creating the test cases manually within the TDE perspective, you will use the Test Data view to enter or review the input values and expected results of all test cases and test steps.

Important: CTE exported values are read-only within the TDE perspective. The cells are insensitive. Switch to the CTE perspective to change such values if necessary (respectively the underlying document).

Icons of the view tool bar

icon	action / shortcut / comment
	Highlights the next failed value.
	Highlights the previous failed value.
	Highlights the next passed value.
	Highlights the previous passed value.
	Highlights the next undefined value.
	Highlights the previous undefined value.

continue next page

icon	action / shortcut / comment
	Highlights all undefined values.

Table 6.138: Tool bar icons of the Test Data view

Viewing the interface of the test object

The Test Data view displays the interface of the test object. On the left side of the Test Data view you see the following interface elements and icons:

Important: Interface elements with the passing direction "Irrelevant" or "Extern" will NOT be displayed within the TDE!

element / icon	means
Inputs and Outputs	Input values are all interface elements that are read by the test object. Output values are written by the test object respectively are the expected results. Within the TIE you determine which values are Inputs and which are Outputs. TESSY tries to find out the default passing directions (input or output) automatically when analyzing the source files.
Globals	Globals are the global variables referenced by your test object. Global variables can contain global static variables and static local variables that are defined within functions.
Parameter	Parameters of the functions of the test object.
Dynamics	Pointer targets, referenced through a pointer of the test object interface.
Return	Return variables.
(white arrow)	The arrow is displayed when an element has child elements. Click on the arrow to expand. If you want to expand all child elements, use the context menu ("Expand all").
(blue dot)	Variable

Table 6.139: Interface elements and icons of the Test Data view

Every variable will be assigned to one of the interface elements described above, e.g. Parameter, Global etc. Initially, the Dynamics section will always be empty.

The colums on the right represent the test steps where the values of the variables are defined.

Please notice the following habits of this view:

- Select a column by clicking on the number of the test step. The selected column is marked in blue (compare figure 6.79).
- Move the mouse pointer over the number of the test step to see the name of the test step within a tool tip (compare figure 6.79).
- Select all values for a variable by clicking on the variable in the left column.
- If you select the icon "Highlight Undefined Values" in the tool bar, all variables that do not contain any data are marked in yellow (compare figure 6.79).

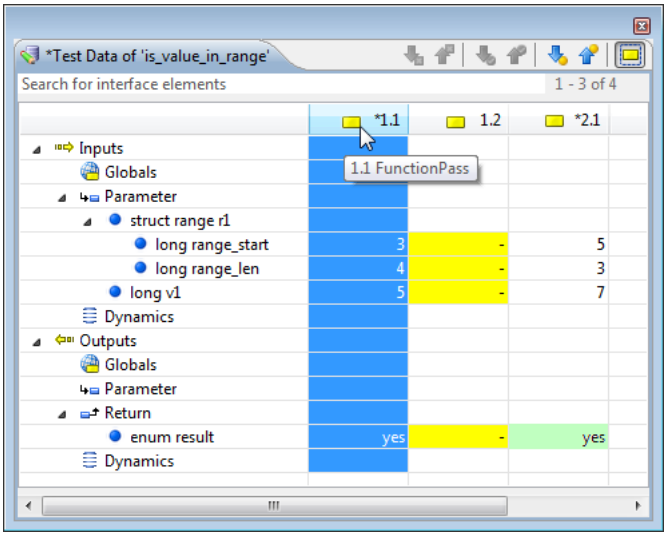


Figure 6.79: Test step 1.1 is selected and undefined values are highlighted in yellow

Entering values

Entering values

Values for interface elements are entered into the cells of the Test Data view. The values will be checked and/or converted when leaving the cell or when changing to any neighboring cell.

Validation of test data values

By default, all imported or manually entered test data values are checked for syntactical correctness, truncated to the type size and optionally formatted. The truncation of values depends on which kind of number format was used:

- Decimal numbers will be checked against the minimum and maximum value of the respective data type. When entering -10 for an unsigned type you will see a value of 0 as test data. If the value is less than the minimum, the minimum value will be used, if it is more than the maximum, the maximum value will be used.
- Hexadecimal and binary numbers will be truncated to the number of bytes available for the respective data type, regardless if the type is signed or unsigned. When entering 0xF11 for an 8 bit type you will see a value of 0x11 as test data. Also when entering a binary 0b1100001111 you will see a value of 0b00001111 as test data.
- Missing leading zeros will be filled up for hexadecimal and binary values. If you enter 0x12 for a 16 bit value, you will see a value of 0x0012 as test data.

After the truncation of the value to the available data type size, it will be formatted according to the data format configured within TIE. Suppose you have an 8 bit signed value with data format "Decimal" and you enter a value of 0xF05: The value will firstly be truncated to 0x05 and then formatted as decimal number so that you will see 5 as test data value.

Important: If you change the data format within TIE, only newly entered test data will be formatted according to the new format. If you want to change the format of the available test data for a certain variable, you need to use the "Convert to Data Format" menu entry within TDE. Make sure the box "Enable Value Checking and Conversion" is checked within the menu "Window" > "Preferences" > "Test interface Settings".

The TDE provides undo/redo functionality for all changes within the Test Data view.

Important: When running the test with undefined values, the initial value passed to the test object depends on the default initialization of the compiler.

Clicking into a cell activates the inline editing mode and you can enter arbitrary values:

→ Click in a cell and press enter.

Now you are in the inline editing mode (editing mode).

→ Enter the value.

You can navigate between the cells with CTRL + cursor left/right.

You can apply the available operations of the context menu to multiple cells depending on the current selection within the Test Data view:

- If you select a single variable of the interface tree, all values of all test steps for this variable will be affected.
- If you select a test step column, all variables of this test step will be affected.
- If you select an array, a struct or a union, all components of this element will be affected.

The current selection is highlighted in blue. You need to select a test step column before right clicking for the context menu, because the right click will not select the test step column.

Defines

Setting defines

→ Double click in a cell or press ENTER to be in the inline editing mode.

→ Press Ctrl + Space.

A window will open with all known defines.

→ Choose your define with a doubleclick.

Enums

Enums

→ Click in a cell.

A dropdown menu will open showing the available enum constants (see figure 6.80.

→ Choose any constant or click into the inline editor field to enter any other suitable value.

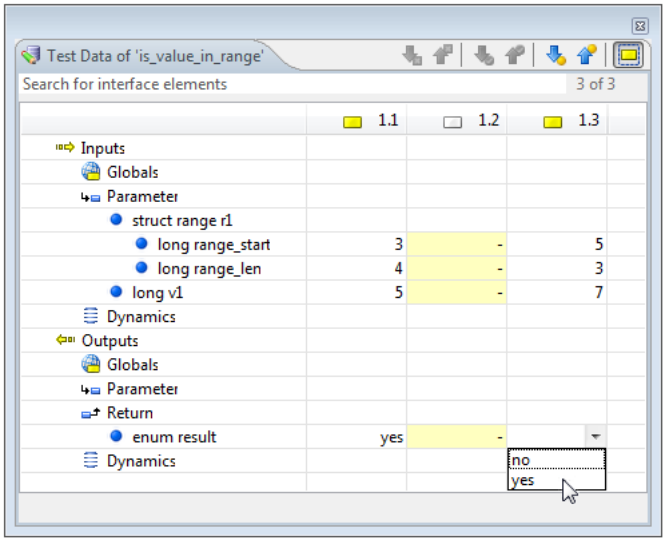


Figure 6.80: Clicking in the cell opens the available enums

Input values

Input values are all interface elements that need to have a defined value at the beginning of the execution of a test object. These values are included in the calculation of the output values during test execution or will be used as a predicate of a condition.

Types of input values

There are three types of input values:

- Global and external variables, used by the test object.
- Function parameters, transferred during function call.
- Dynamic objects: They represent pointer targets, referenced through a pointer of the test object interface (they are not really dynamic, variables will be created for each one within the generated test driver).

Expected values

Expected values are the calculated results regarding the input values for the test object after test execution. TESSY will compare both expected and actual values after test execution. The result will be either failed or passed.

Important: The values are compared with the evaluation mode equal (==). To change the evaluation mode refer to section Entering evaluation modes.

There are three types of expected values:

Types of expected values

- Global or external variables: They can always be expected values as they have a valid value after execution of a test object.
- The return value of a function.
- Dynamic objects: Referenced through a pointer that is either an input or expected value, dynamic objects continue to be accessible after execution of a test object. They can therefore also be expected values, just like global variables.

Initializing test data

You can initialize all interface variables of a test step at once.

→ Rightclick the variable and choose "Initialize Test Data..." from the context menu.

The "Initialize Values" dialog opens.

You can use following options:

option	**means**
Pattern	a pattern in hexadecimal format, i.e. 0xff
Value	a specific value only
Random	a range of generated values for the initialization. The random values will adhere to the min/max limits of each interface variable type
Ignore values	all input and expected values will be set to "*none*"
Initialize all array elements	all array elements will be initialized. Otherwise only visible array elements will be initialized

Table 6.151: Options of initializing values

The following table shows the initialization values for certain data types:

type	contents
Integer	*0x00000000* i.e. if *0x42* is entered as pattern, all int variables will be initialized with *0x42424242*.
Float	*0.0*
Char	the pattern or numerical value
Struct	all sub components are initialized according to their type
Union	the first sub component is initialized as active component
Enum	the first enum constant is used as initialization value
Array	all array elements are initialized according to their type if option "Initialize all Array Elements" is used
Pointers	Pointers are initialized with *NULL* provided that they do not point to dynamic objects

Table 6.152: Initialization values for data types

Entering evaluation modes

Using the evaluation mode allows to specify how to compare the actual value (calculated during the test run) with your specified expected value.The evaluation mode together with the expected value will be used to process the test results after the test run.

The default evaluation mode is equal (==).

Evaluation modes

To enter another evaluation mode:

→ Click in a cell.

→ Enter the desired evaluation mode within the inline editor mode (see figure 6.81).

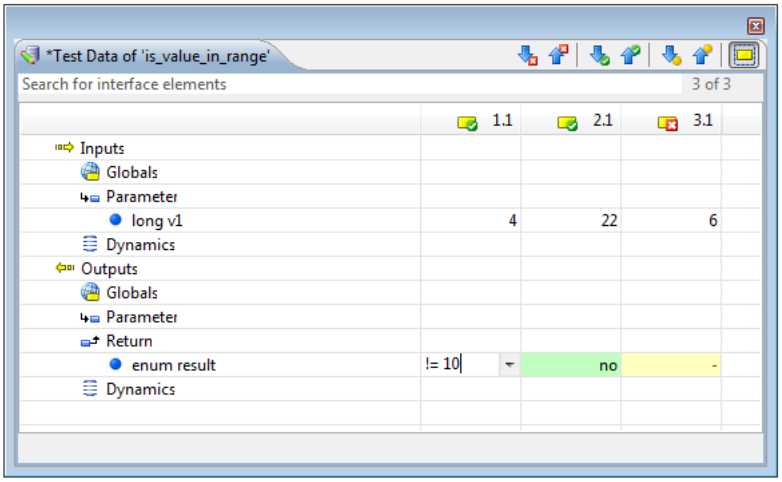


Figure 6.81: Entering evaluation mode "unequal" within the inline editor

The following evaluation modes are available:

evaluation mode	**written as**	**means**
equal	==	Checks the expected value and actual value for Equality. **This is the default setting.**
unequal	!=	Checks the expected value and actual value for inequality.
greater	>	Checks if the actual value is greater than the expected value.
less	<	Checks if the actual value is less than the expected value.
greater or equal	>=	Checks if the actual value is greater or equal to the expected value.
less or equal	<=	Checks if the actual value is less or equal to the expected value.
range	[1:10]	Checks if the actual value is within a range, here: range 1 to 10.
deviation	100 +/- 1 100 +/- 1%	Checks if the actual value equals the expected value but takes into account a deviation value. The deviation can either be an absolute value or a percentage, i.e. the following expected values would yield OK: 99, 100, 101.

Table 6.154: Evaluation modes

Ignoring values for a test step

By default, values have to be assigned for all variables with passing directions "IN" or "INOUT". It can be useful to not overwrite a value calculated in the last test step. In this case you can use the special value "*none*":

→ Rightclick a value and choose "Ignore Value" within the contextmenu.

Pointers

→ Rightclick the pointer value cell to open the context menu.

The context menu offers the following possibilities to assign a value for a pointer:

option	**means**
Set Pointer NULL	The value of the selected pointer will be set to NULL. The text box will be filled with NULL.
Set Pointer Target	You can select another interface element or a component of a structure or union and assign its address to the pointer. The cursor will change, when you move the mouse pointer over a variable: ⌖ The object type fits the pointers target type. You can assign the pointer. ⊘ The object type does not match the pointers target type. You cannot assign the pointer. When you click on an element, the variable name of that element will be entered into the field next to the pointer. During test execution, the address of the variable will be assigned into the input field of the pointer.
Create pointer target value	Allows to create a new object as target object for the pointer. The address of the object will be assigned to the pointer. The type of the created object depends on the target type of the pointer. → Choose "Create Pointer Target Value" from the context menu. → Enter a valid C identifier as name for the new target object. → Click "OK". A new target object will be listed in the dynamic objects section of the TDE.

continue next page

option	means
Array as Target Value	It is also possible to create an array as target value using the Dimension option of the Create Pointer Target dialog: ➞ Tick the check box for" As Array of Size" to enter an appropriate size into the input field. ➞ Click "OK". The name of the new object appears in the input field of the pointer value. TDE will create an array of the pointers target type. The pointer will point to the first array element. Within the Dynamics section, you will see the newly created target object. You can enter values, like for every other interface element.

Table 6.157: Value assignments for pointers

Static local variables

Variables defined as static local variables within the test object or called functions can also be tested. Since such variables are not accessible from outside the location where they are defined, TESSY instruments the source code and adds some code after the variable definition to get a pointer to the memory location of the variable. All static local variables can only be accessed after the code containing the variable definition has been executed. You need to keep this in mind when providing input values or checking results for such variables. The following restrictions apply for static local variables:

- The first time when the code of a static local variable definition is executed, the variable will get the initialization value assigned from the source code. It is not possible to set the initial value from TESSY. You need at least one test step to initialize the variable by executing the definition code. The next test step can then supply an input value for the variable.
- The same applies for expected values: If the source code of the variable definition has not been executed, the result value of the respective variable is not accessible and will be displayed as *unknown* in this case. This situation may arise when the variable definition is located within a code block which has not been executed, e.g. within an if statement block.

6.7.8 Test Definition view

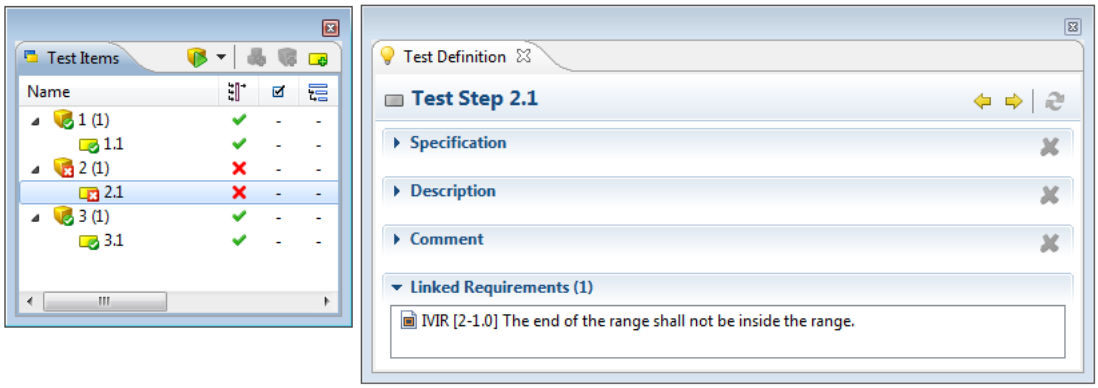


Figure 6.82: Test Definition view within TDE with linked requirement

The Test Definition view displays the test case specification, the optional description and linked requirements of the current test case in individual input fields. The test case specification should enable the tester to provide concrete input values and expected results.

The Test Definition view is context sensitive! To display the specifications, definitions and requirements for a test case:

→ Select a test case within the Test Items view (see figure 6.82).

Important: The contents are not editable if the test cases have been created and exported using the CTE!

6.7.9 Prolog/Epilog view

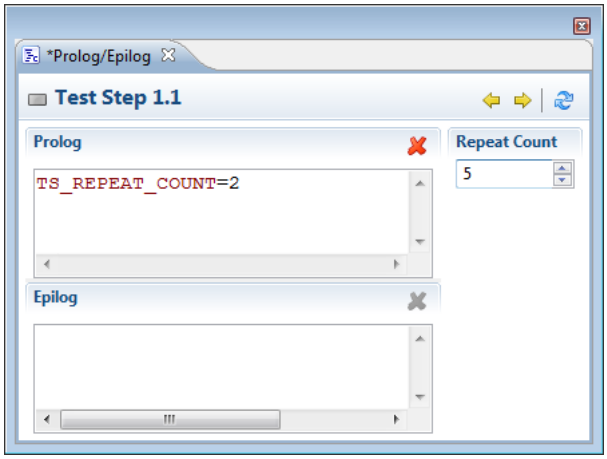


Figure 6.83: Prolog/Epilog view

Within the Prolog/Epilog view you can specify usercode that will be executed at a certain point in time during the test execution. The C part of the usercode will be integrated into the test driver and executed at the places specified. The following figure outlines the call sequence of the usercode parts.

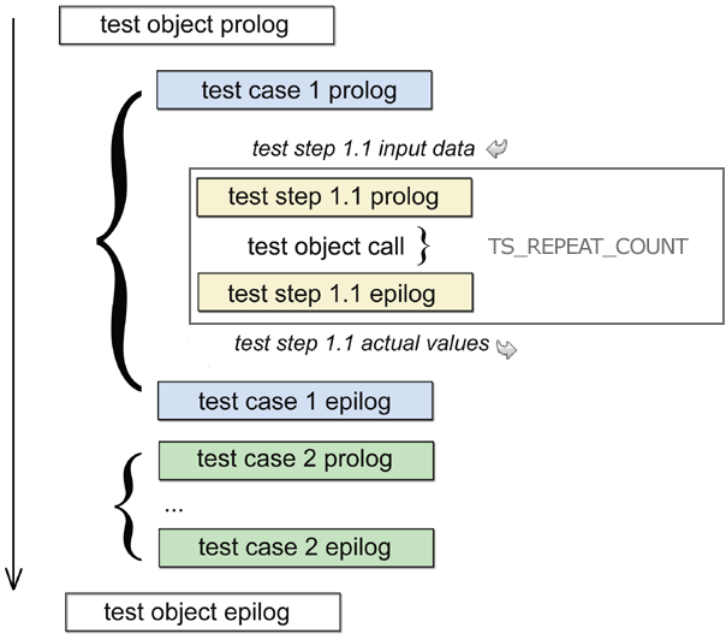


Figure 6.84: Call sequence of the usercode parts

The figure shows the interaction of the usercode sections with the assignment of test data provided within TDE and the result values that are saved into the test database and evaluated against the expected results.
During the test object call, the code specified for the stub functions (if any functions are called from your test object) will be executed depending on the code logic of your test object.

Example

Have a look at figure 6.83 Prolog/Epilog view in the beginning of this section. The test step 1.1 prolog contains the code TS_REPEAT_COUNT=2, and the Repeat Count for this prolog/epilog section was set to 5.
The whole prolog / test object call / epilog sequence wil be repeated five times and the test object will be called twice in every repetition of this loop. Since there are 5 loops, the test object will be called 10 times in total.

Entering C code

The Prolog/Epilog view provides a popup menu containing variables for convenient editing.

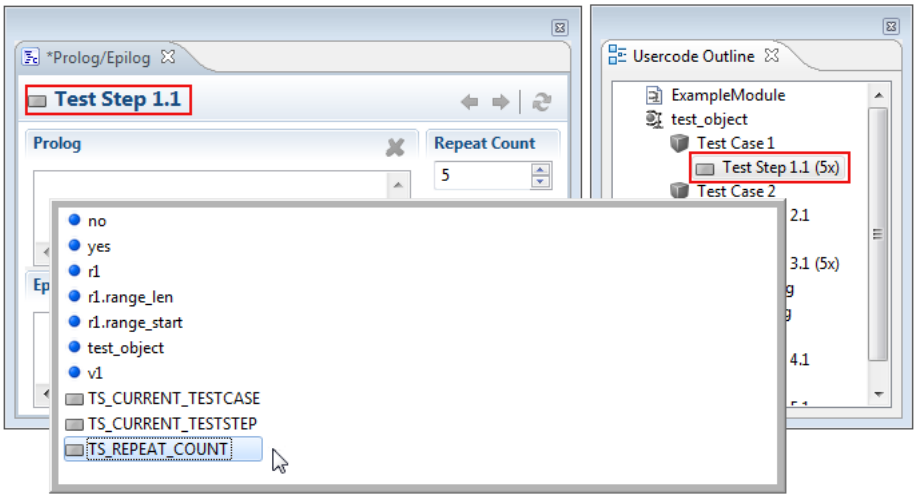


Figure 6.85: Prolog/Epilog functions

To show this menu,

→ use the Usercode Outline view to mark the test case or test step for which you want to set the usercode.

→ Click into the Prolog or Epilog section of the Prolog/Epilog view and enter the usercode.

→ Press CTRL + Space or type the first letters and press CTRL + Space.
The popup menu appears (see figure 6.85), showing all available names respectively the filtered list according to the characters you already typed.

Important: TS_REPEAT_COUNT is only usable within the prolog section!

To edit the prolog/epilog for a test case/test step,

→ Use the Usercode Outline view to navigate and select a test case or test step from the tree.

→ Enter the code within the Prolog/Epilog view.
A new node will automatically appear at the corresponding place in the tree (see figure 6.86).

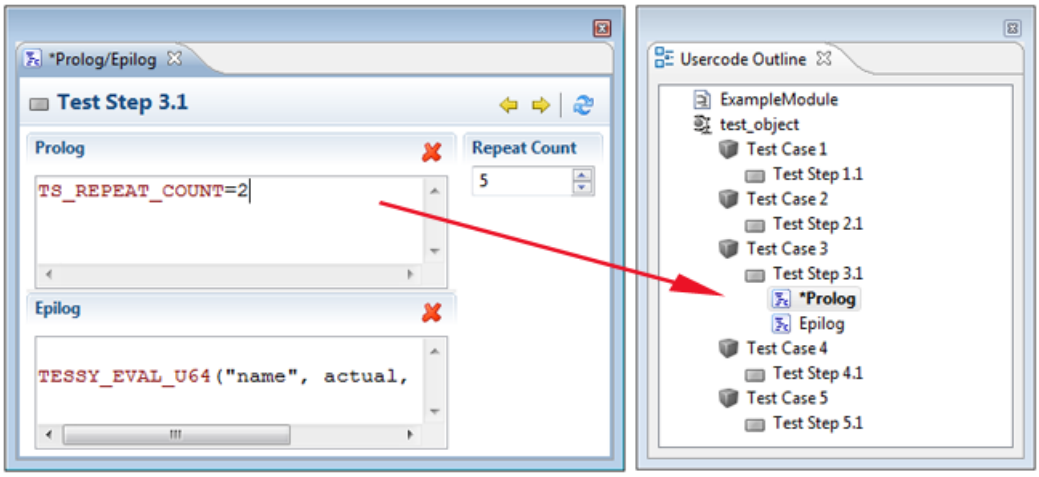


Figure 6.86: Editing C code

Using evaluation macros

Within the test step epilog or within stub functions, You can evaluate any variable or expression using the evaluation macros. These predefined macros allow to check an expression against an expected value. The result is stored within the test report like the evaluation of normal output variables of the test object.
Evaluation macros can only be used within the following Usercode sections:

- test step epilog
- stub function code

Evaluation macros

A popup menu contains all available interface variables and symbolic constants for convenient editing as well as the available evaluation macros, e.g. TESSY_EVAL_U8 for unsigned character values:

→ Press CTRL + Space.

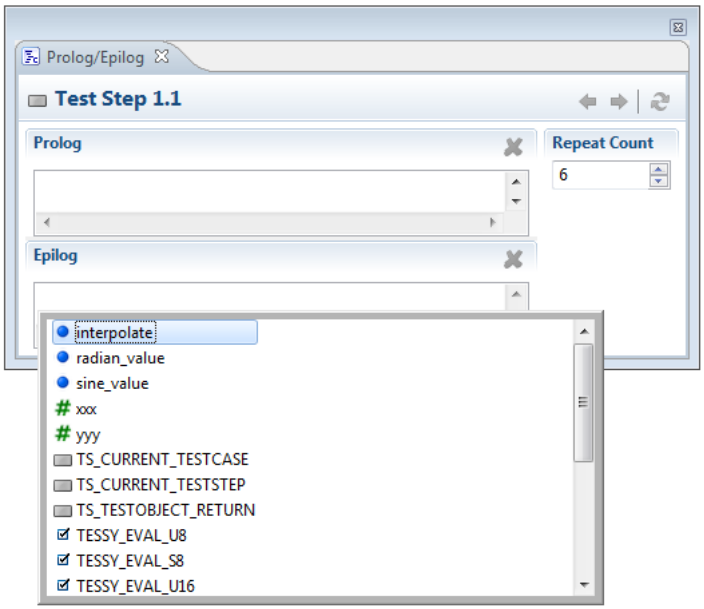


Figure 6.87: Call the popup menu by pressing CTRL + space

The only difference of the evaluation macros is the type of argument for the actual and expected value, see table below for a description of the available types.

evaluation macro name	C type
TESSY_EVAL_U8	unsigned, 1 byte
TESSY_EVAL_S8	signed, 1 byte
TESSY_EVAL_U16	unsigned, 2 byte
TESSY_EVAL_S16	signed, 2 byte
TESSY_EVAL_U32	unsigned, 4 byte
TESSY_EVAL_S32	signed, 4 byte
TESSY_EVAL_U64	unsigned, 8 byte
TESSY_EVAL_S64	signed, 8 byte
TESSY_EVAL_FLOAT	float
TESSY_EVAL_DOUBLE	double
TESSY_EVAL_LONGDOUBLE	long double

Table 6.164: Available types of evaluation macros

operator	meaning
==	equal
!=	unequal
<	less
>	greater
<=	less or equal
>=	greater or equal

Table 6.165: operators of evaluation macros

Each invocation of an evaluation macro results in an additional entry within the test report. All evaluation macros will be added to the list of actual/expected values of the current test step. The results will be displayed within the Usercode Outline view and the Evaluation Macros view.

It is possible to format the output of the evaluation macros as binary value, decimal or hexadecimal (default setting) by appending one of the following format specifiers at the end of the evaluation macro name:

binary	`%bin`, e.g. "`Value printed as bin%bin`"
decimal	`%dec`, e.g. "`Value printed as dec%dec`"
hexadecimal	`%hex`, default setting.

6.7.10 Usercode Outline view

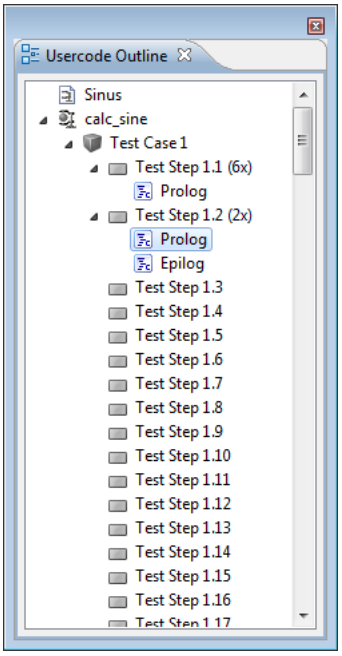


Figure 6.88: Usercode Outline view

The Usercode Outline view displays the usercode that will be executed at a certain point in time during the test execution and that you just defined in the Prolog/Epilog view.

6.7.11 Plots view

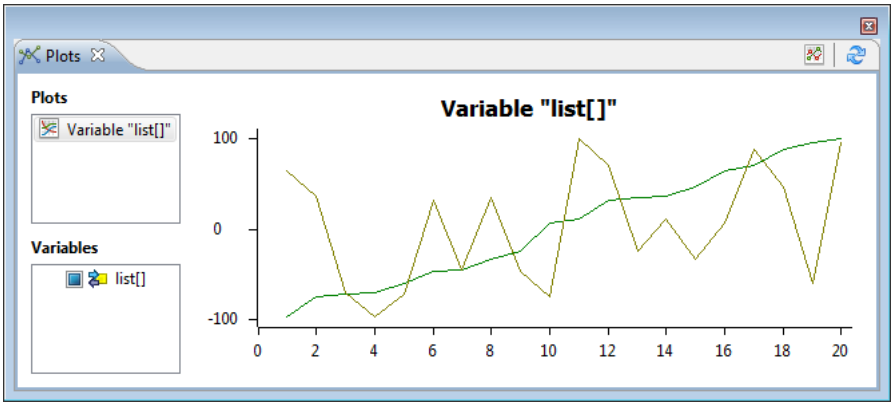


Figure 6.89: Plots view

The Plots view visualizes graphically the test results.

Important: If you use other evaluation modes than equal (e.g. <, <=, >, >=, !=, [Range]), it is **not** possible to see the expected values. To see expected values use the evaluation mode == (equal). See section 6.7.7 Entering evaluation modes.

Icons of the view tool bar

icon	action / shortcut / comment
	Shows the symbols for all series, i.e. shows dots for each distinct value on the value curves.
	Reloads the plot data.

Table 6.167: Tool bar icons of the Plots view

Entering plots

To enter a new plot:

→ Rightclick in the Plots pane.

→ Select "New Plot" in the context menu (see figure 6.90).

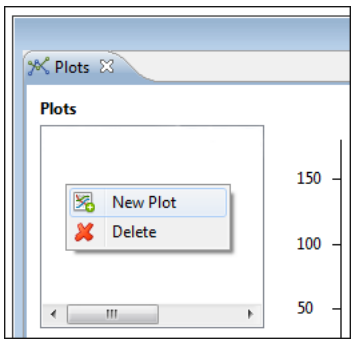


Figure 6.90: Creating a new plot with the context menu

The New Plot window will be opened (see figure 6.91). It will show you the possible items for the plot.

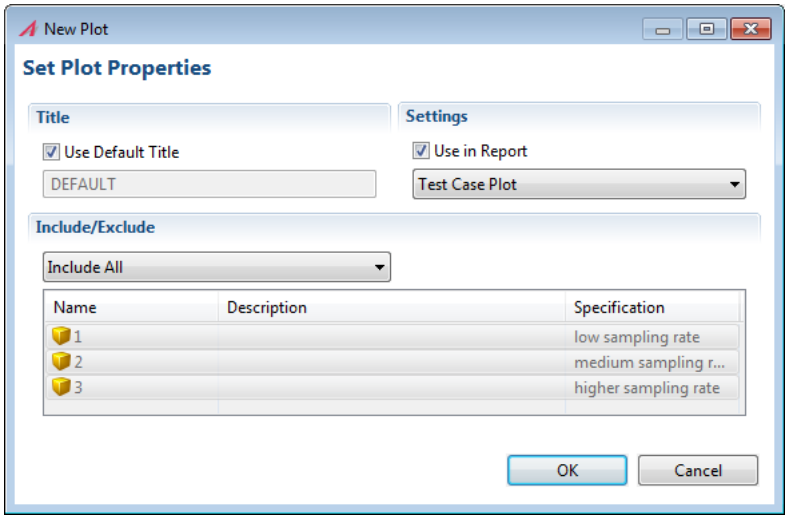


Figure 6.91: Creating a new plot

Setting plot properties

To set the properties for the plot:

→ Under "Title" choose the default title or enter an own title. The default title will be build from the names of the variables used within the plot.

→ Under "Settings" select, if you want to see the plots grafic in the report and which kind of grafic: You can select between a test case plot, a test step plot or a whole array plot.

A **test case plot** spanns over all values of all test cases of the selected variables.
A **test step plot** provides one curve for each test case spanning over all values of the test steps of this test case. This requires at least two test steps for each test case to define a valid curve.
An **array plot** creates plots for array type variables. There will be one curve spanning over the array values for each test step.

→ Under "Include/Exclude" select, which test cases you want to select: With Shift + selection you can select test cases for include or exclude.

Important: Be careful with selecting "Include Selected": If you add new test steps afterwards, they will not be included automatically! In that case it might be more useful to select "Exclude Selection".

→ Click "OK".

To add variables to the plot:

Adding variables to plot

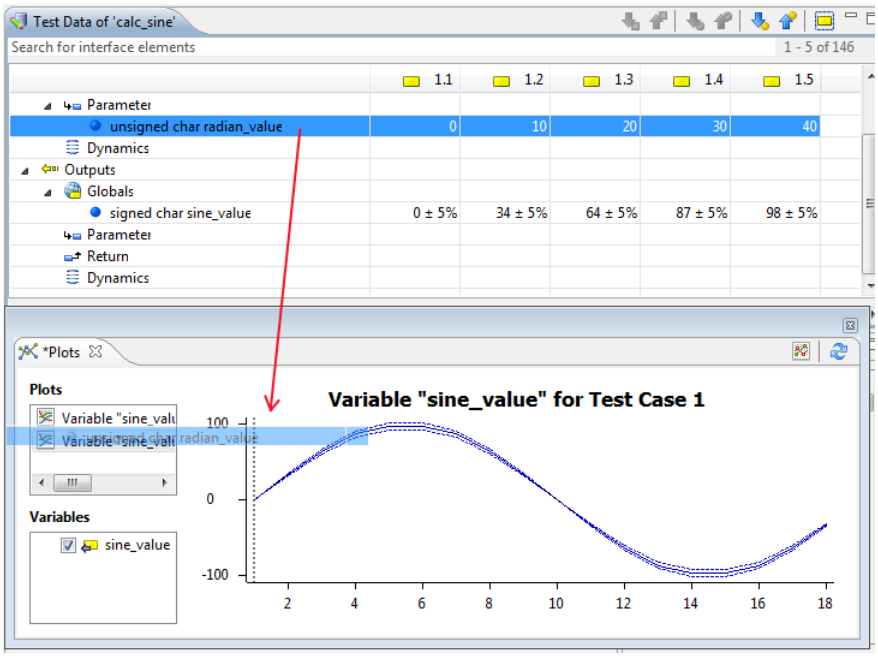


Figure 6.92: Adding variables to a plot

→ Drag and Drop the variables to the plots (see figure 6.92.).

You can add several variables. In our example we added two variables (see figure 6.93):

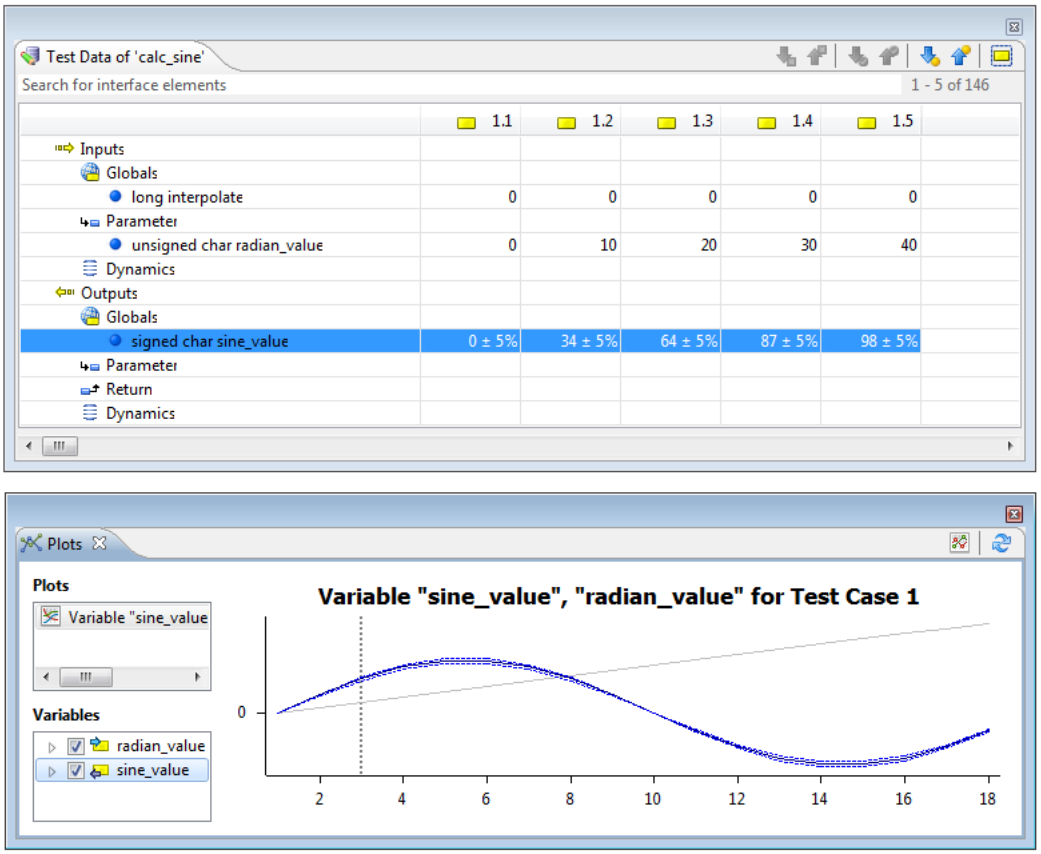


Figure 6.93: Plots view with a plot and two variables

Important: When the values of variables differ significantly, it may not be possible to scale them usefully within one plot. In this case you should use individual plots for each variable instead.

You can select a variable, or uncheck a box to see only the checked plot (see figure 6.94).

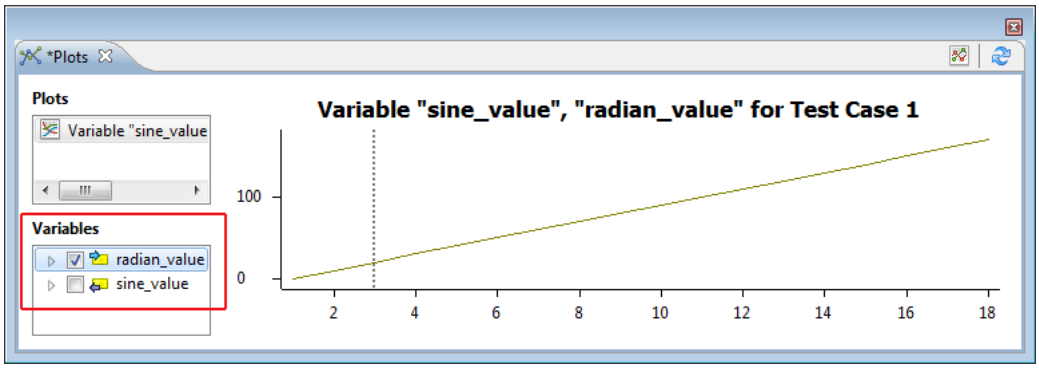


Figure 6.94: Uncheck variables for a better overview

If you mark the upper or lower value, the curve will be highlighted in blue (see figure 6.95).

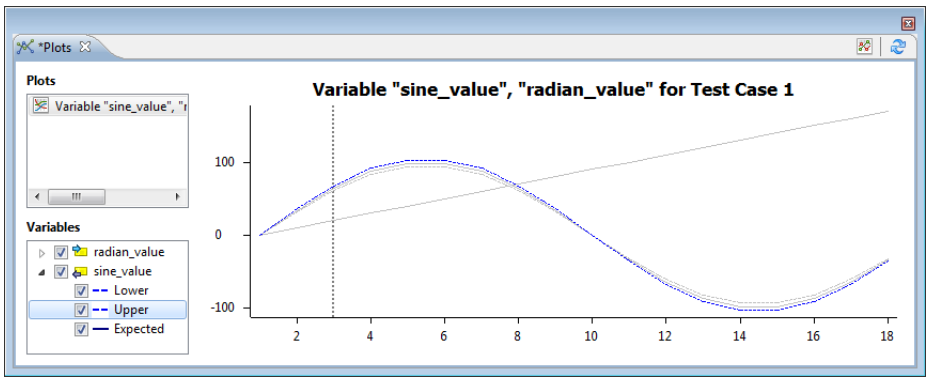


Figure 6.95: Highlighted upper value for variable sine_value

It is possible to save the view as a .jpg oder .png-file:

- → Righclick on the Plots view and select "Save As...".
- → Select where you want to save the file (default: "projects" > "samples").
- → Enter a name and select the kind of file.
- → Click "Save".

All these settings will be saved and the plots within the test report will be created as currently displayed within the plot view!

6.8 CV: Analyzing the Coverage

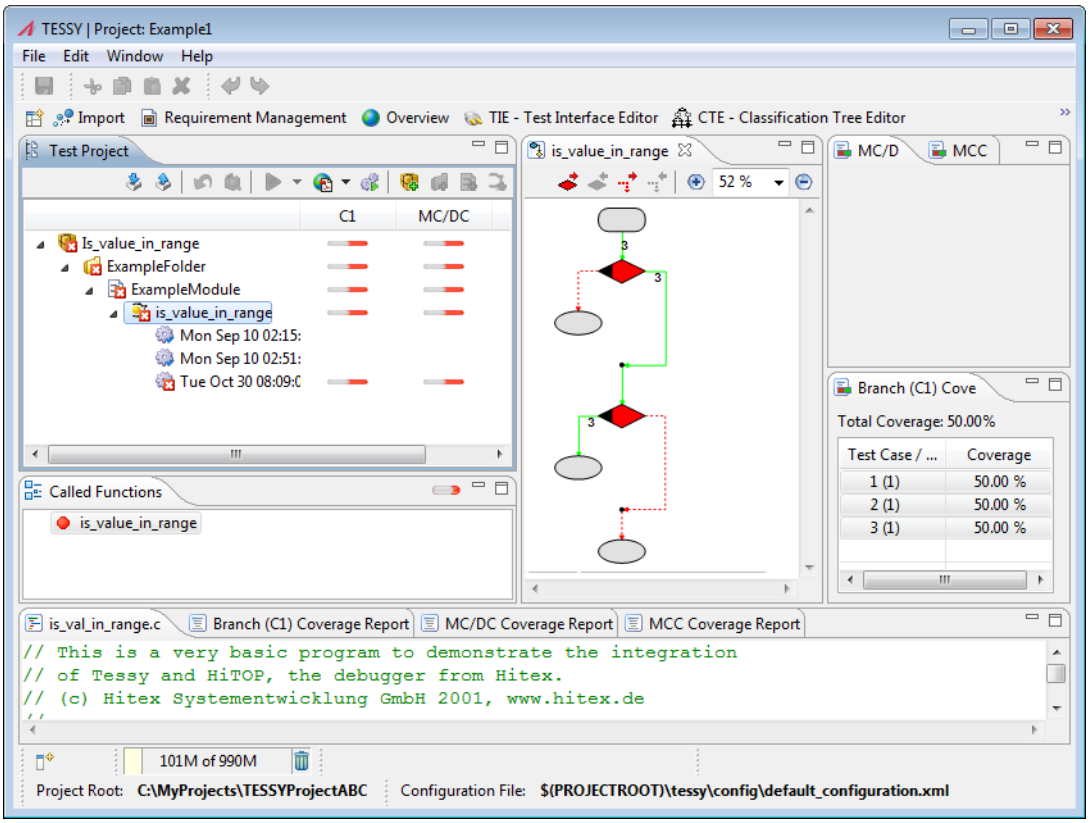

TESSY | Project: Example1
File Edit Window Help
Import Requirement Management Overview TIE - Test Interface Editor CTE - Classification Tree Editor
Test Project
C1 MC/DC
Is_value_in_range
ExampleFolder
ExampleModule
is_value_in_range
Mon Sep 10 02:15:
Mon Sep 10 02:51:
Tue Oct 30 08:09:0
Called Functions
is_value_in_range
is_value_in_range
52 %
MC/D MCC
Branch (C1) Cove
Total Coverage: 50.00%
Test Case / ... Coverage
1 (1) 50.00 %
2 (1) 50.00 %
3 (1) 50.00 %
is_val_in_range.c Branch (C1) Coverage Report MC/DC Coverage Report MCC Coverage Report
// This is a very basic program to demonstrate the integration
// of Tessy and HiTOP, the debugger from Hitex.
// (c) Hitex Systementwicklung GmbH 2001, www.hitex.de
101M of 990M
Project Root: C:\MyProjects\TESSYProjectABC
Configuration File: $(PROJECTROOT)\tessy\config\default_configuration.xml

Figure 6.96: Perspective CV - Coverage Viewer

TESSY provides the following coverage measurements:

- C1 branch coverage
- MC/DC modified condition / decision coverage
- MCC multiple condition coverage

The Coverage Viewer (CV) displays the results of the coverage measurement of a previously executed test. The available information displayed and the sub windows shown within the CV depend on the coverage options selected during the test run.

The CV will be updated with the coverage information of the currently selected test object whenever you switch to the CV or when you select another test object within the Test Project view.

Refer as well to the application note "Coverage Instrumentation" for more information and usage of coverage analysis in TESSY ("Help" > "Documentation").

To select the desired coverage measurement for the test run:

→ In the tool bar of the Test Project view click on the arrow next to the Execute Test icon and select "Edit Test Execution Settings . . . ".

→ In the following dialog choose the desired coverage measurements (see figure 6.97).

→ Click on "Execute".

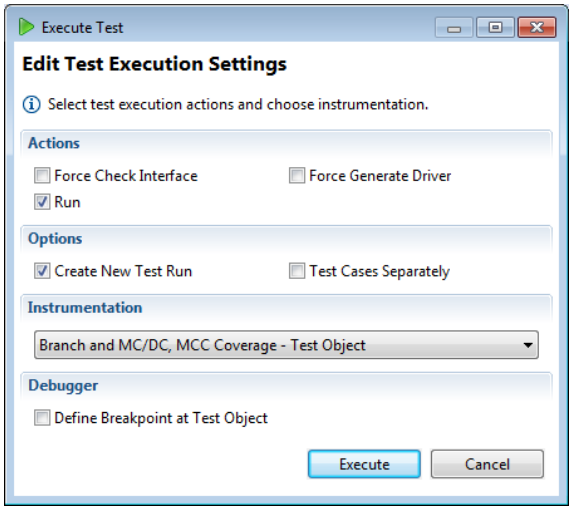


Figure 6.97: Selecting Branch and MC/DC Coverage for test run

The maximum information being displayed will be available with the instrumentation option C1/C2 - test object and called functions. The applied coverage depends on which coverage measurements are enabled. In case of C2 this may be either MC/DC and/or MCC coverage.

6.8.1 Structure of the CV perspective

pane	**location** (default)	**function**
Test Project view	upper left	Same view as within the Overview perspective.
Called Functions view	middle left	Contains the test object itself and the functions called from the test object.
flow chart view	upper middle	Displays the graphical representation of the control structure of the currently selected function (only displayed when a coverage mesurement was selected for a test run!).
C1-, MC/DC-, MCC-Coverage views	upper / middle right	Displays the C1, MC/DC and MCC coverage results.
code view	lower pane	Displays the source code of the currently selected function (and highlighting selected decisions/branches).
C1-, MC/DC- and MCC-Report views	lower pane	Displays the ASCII based summary reports for C1, MC/DC and MCC coverage.

Table 6.174: Structure of CV

6.8.2 Test Project view

6.1.2 Test Project view

The Test Project view displays your test project which you organized within the Overview perspective.

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

6.8.3 Called Functions view / Code view

The Called Functions view contains the test object and displays the functions called from the test object. By clicking on a function of the test object, you can review the source code within the Code view and see the code structure within the flow chart view.
When selecting an element within the Flow Chart view, the respective source code lines will be marked within the Code view.

6.8.4 Flow Chart view

The flow chart view displays the code structure and the respective coverage in graphical form.

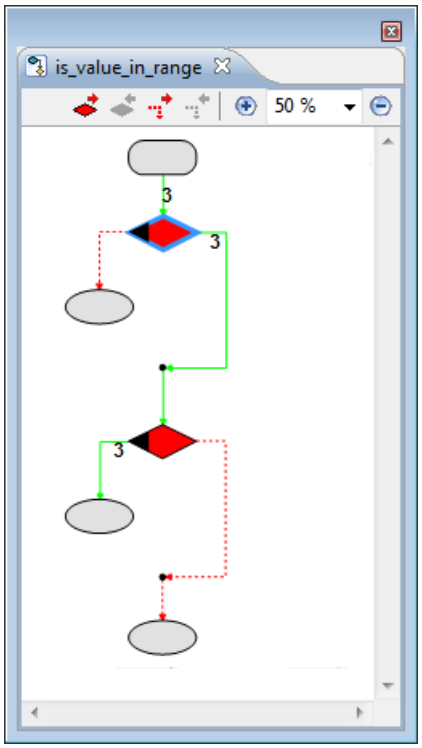


Figure 6.98: Flow Chart view

Icons of the view tool bar

icon	action / shortcut / comment
	Searches the next uncovered decision.
	Searches the previous uncovered decision.
	Searches the next unreached connection.
	Searches the previous unreached connection.
	Zooms out.

continue next page

icon	action / shortcut / comment
	Zooms in.

Table 6.175: Tool bar icons of the Test Items view

Viewing functions

To display a flow chart of a function:

→ Click on a function within the Called Functions view.

The code structure of the function will be displayed in the flow chart view.

Zoom in or out using the tool bar icons or the entries from the chart menu.

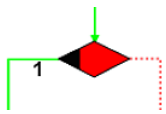
Viewing Functions

Within each flow chart, you will see the branches of the function being displayed in green and red colors, which indicates whether the respective decision has been fully covered or the respective branch has been reached:

green = 100 % coverage (MC/DC and MCC)
red = less than 100 % coverage: at least one condition combination was not executed

If no C2 coverage has been selected for the last test execution, the decision elements remain grey, but they are still selectable in order to find the respective line of code in the source code view.

The following elements are displayed within the flow chart of the CV:

	**if decision** The if branch on the left side was not reached, the else branch on the right side was reached three times. The decision was not fully covered.

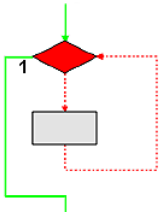
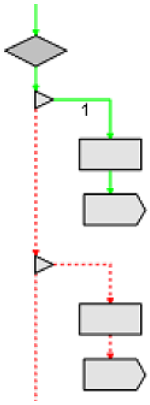
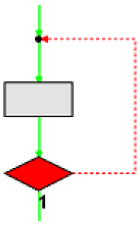
	**for or while loop** The loop body was not reached, instead the exit out of the loop was executed once. The loop decision was not fully covered.
	**switch statement** The first case branch was reached once, the second case branch and the default branch were not reached.
	**do while loop** The loop body was only reached once (without repeated execution of the loop body branch) and the exit branch was reached. The loop decision was not fully covered.

Table 6.177: Elements of the Flow Chart view

Selecting Elements

You can select decisions, branches and code statement elements within the flow chart. The respective code section will then be highlighted within the source code view. Since not all connection lines within the flow chart are branches in terms of the C1 branch definition, some of the connection lines may not be selectable.

If an element (e.g. the ? operator and statements containing boolean expressions) also appears in green or red, the element contains sub flow charts that can be visualized with a right click on the respective element. CV will insert a new tab for the condition.

Searching for uncovered decisions and unreached branches

The CV provides search functionality for decisions and branches that are not fully covered respectively reached through all the executed test cases. The decisions and branches are already marked in red, but the search function can assist in finding all uncovered decisions or unreached branches.

→ Select the respective icon from the tool bar (i.e.).

The chart will change into the search result mode, marking the found element blue.

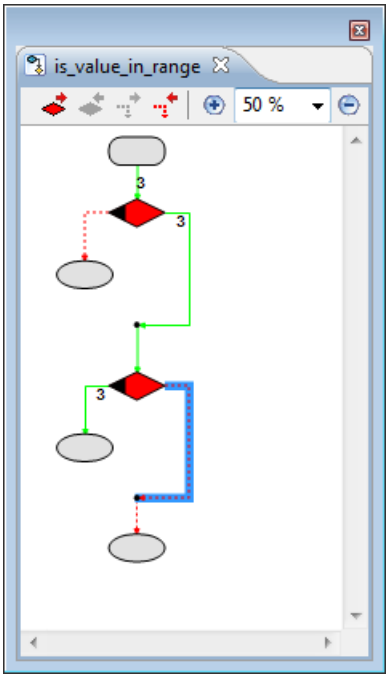


Figure 6.99: Unreached connection is marked blue

6.8.5 Branch (C1) Coverage view

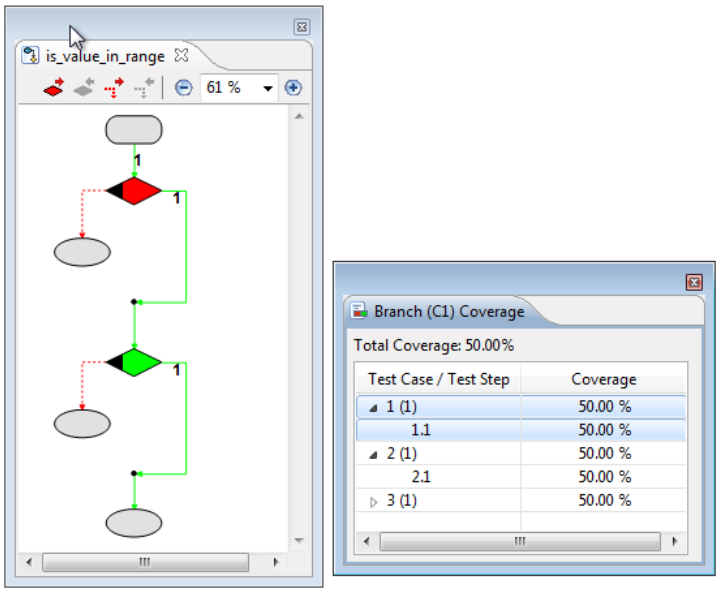


Figure 6.100: Branch coverage

The C1-Coverage view displays the branch coverage for each individual test case and test step as well as the total coverage for all test cases (and test steps). The coverage information in this view is calculated for the selected function within the Called Functions view.

If you selected only the C1 coverage instrumentation for test execution, you will see only the C1 branches marked in red and green within the flow chart.

If you select individual test cases or test steps within the test case list, the respective branches covered by those test steps will be marked within the flow chart. This allows finding out the execution path of the selected test step. By selecting multiple test steps, review the resulting cumulated branch coverage within the flow chart. The total coverage number will also be updated with the C1 branch coverage for the selected test cases / test steps.

Coverage percentage

The coverage percentage is the relation between the total numbers of branches of the currently selected function compared to the number of reached branches. This coverage calculation includes the currently selected test cases and test steps within the test case / test step list. By default, all test cases are selected when opening the CV.

6.8.6 MC/DC Coverage view

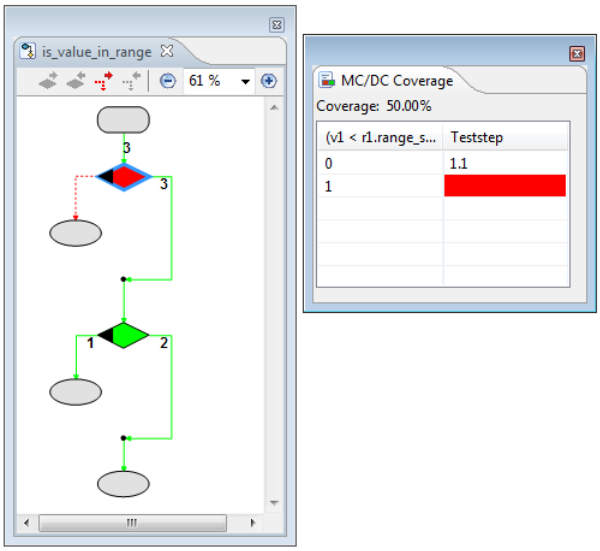


Figure 6.101: Decision coverage

The MC/DC-Coverage view displays the coverage of the currently selected decision within the flow chart view (see figure 6.101). If no decision is selected (as initially when starting the CV), the MC/DC Coverage view is empty.

When selecting a decision, the respective combination table according to the MC/DC coverage definition will be displayed within the MC/DC-Coverage view like shown below. The example decision consists of 4 atoms. All condition combinations are fully covered by the sample test steps.

The combination table contains all atomic conditions of the decision. The conditions are the basic atoms of the decision which remain after removing the **or**, **and** and **not** operators from the decision. TESSY calculates the MC/DC set of true/false combinations of the condition atoms that fits best to the test steps executed during the test run. The last table column contains the test step that caused the execution of the decision with the true/false combination of the respective table row. If one or more of the condition combinations were not reached during the test run, the test step column of those rows will be marked in red.

Selecting Decisions

→ Select a decision by clicking on the respective control flow element within the flow chart view.

The code fragment will be marked within the source code view.

The decisions are either green or red depending on the degree of coverage. If no C2 coverage information is available (i.e. when you ran the test without C2 instrumentation selected), the decisions within the flow chart will appear in grey and the MC/DC-Coverage view will not be available (N/A).

6.8.7 MCC Coverage view

Refer to the description of the MC/DC Coverage view. The only difference is the calculation according to the definition of the MCC coverage.

6.8.8 Coverage Report views

There are up to three coverage reports available depending on the instrumentation mode selected for test execution. They contain the summarized coverage information of the last test execution:

- The C1 coverage report contains some meta information (e.g. number of branches, reached branches, total branch coverage) and the source code of the test object.
- The MC/DC coverage report lists all decisions of the test object code including the coverage tables with the respective MC/DC condition combinations.
- MCC coverage report also lists all decisions of the test object code including the coverage tables with the respective MCC condition combinations.

6.9 IDA: Assigning interface data

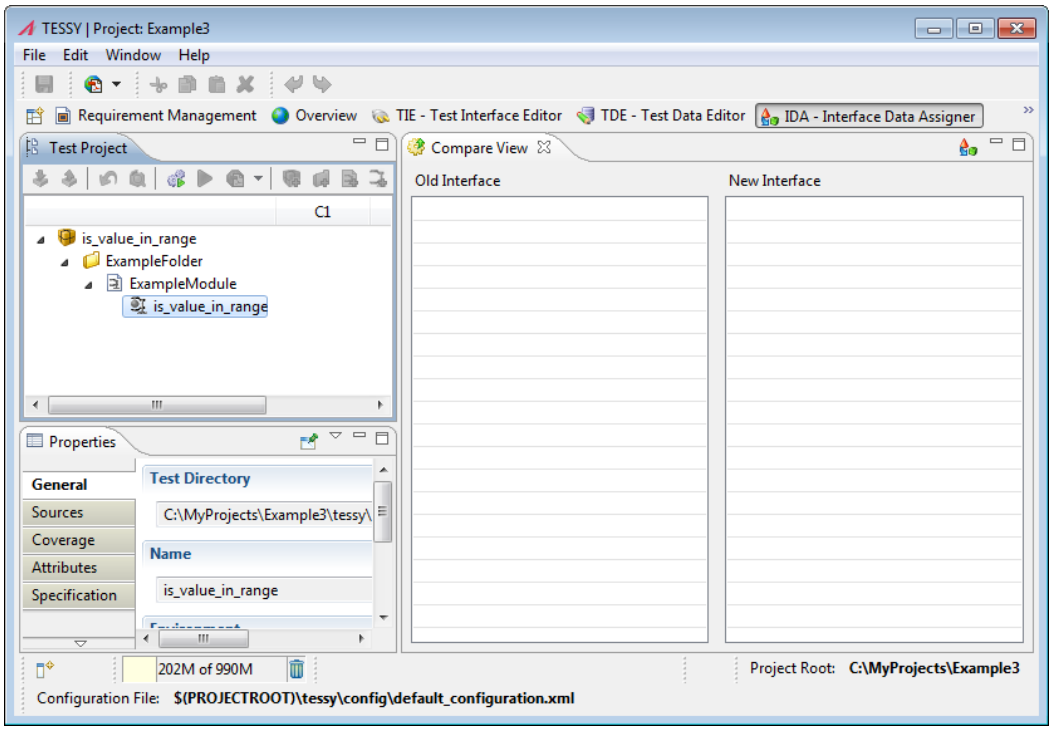


Figure 6.102: IDA perspective

For coherent testing it is essential to realize changes of the interface of test objects and to re-execute previously passed tests to assure that any changes of the source do not cause the previous passed tests to fail. This is often summed up with the keywords "regression testing".
If the interface of the test object changes, TESSY will indicate the changes with specific test readiness states. With the Interface Data Assigner (IDA) you can assign the elements of a changed (new) interface to the elements of the old one and proceed a reuse.

To appropriately react to changes, the test system needs to know the structure of the interface. TESSY examines the source files and determines the functions and their interfaces by analyzing the source code. This information is stored in a special database and can be retrieved any time. This information enables the representation and adjustment of the interface structure independent of the test data or expected values.

All settings on the interface page, e.g. the definition of external variables or stub functions, are specified separate from the data required for the test. This separation of interface information and its test data achieves a clear distinction between structure and data and offers an enormous advantage when handling changes in the source code. On the one hand the test of the interface is possible, indicating changes in the first place. On the other hand, if a change occurs usually only a few elements of the interface of a function under test are changed.

Source files are often modified extensively during the later stages of development, thus making it necessary for a new test to take place, e.g. a regression test. This can alter the interface of an existing test object. In order to allow you to reuse test data that has already been entered, the Interface Data Assign Editor IDA allows the assignment of old interface elements to new ones. This way, old test data can be reused automatically.

6.9.1 Structure of the IDA perspective

pane	**location** (default)	**function**
Test Project view	upper left	Displays your test project. For editing your test project switch to the Overview perspective.
Properties view	lower left	Displays the properties of your test project, e.g. sources to the test object.
Compare view	right	Displays the interface and the changes of the interface. You can assign the changes by drag & drop.

Table 6.181: Structure of the IDA perspective

6.9.2 Test Project view

The Test Project view displays your test project which you organized within the Overview perspective.

6.1.2 Test Project view

Important: We recommend to do any changes of the test project structure within the Test Project view of the Overview perspective. The view layout of this perspective is optimized for this purpose!

6.9.3 Properties view

6.1.3 Properties view

The Properties view displays all the properties which you organized within the Overview perspective. Most operations are possible.

For changing a source switch to the Properties view within the Overview perspective.

6.9.4 Compare view

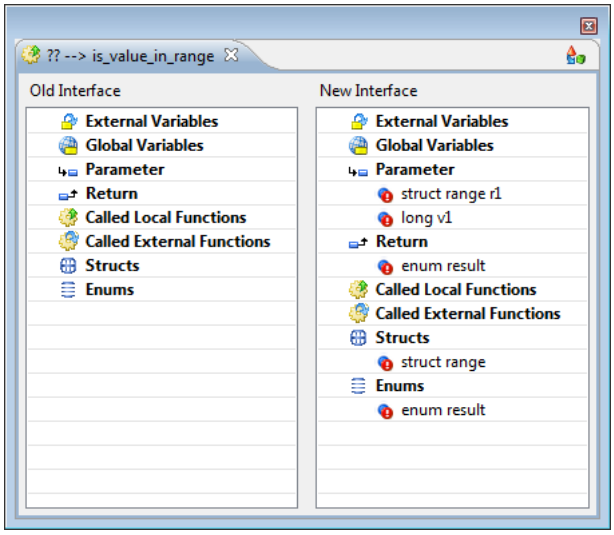


Figure 6.103: Compare view

Status indicators

indicator	status / meaning
	The test object has **changed**. You see these test objects, but there is no operation possible. You have to start a reuse operation.
	The test object is **newly** available since the last interface analysis. You have to add test cases and test steps and enter data for a test.
	The test object has been **removed** or renamed. You still see these test objects, but there is no operation possible. You have to assign this test object to any other and start the reuse operation.

Table 6.182: Status indicators of the Compare view

Comparing interfaces and assigning changes

Within the Compare view you can see the old interface of our test object and the new one. The red exclamation mark within the new interface indicates the need to assign this interface object before starting the reuse.
The title of the view shows the old name versus the newly assigned name.

To assign changes:

→ Use the context menu or just drag and drop from the left side (see figure 6.104).

The red exclamation mark turns to a green check .

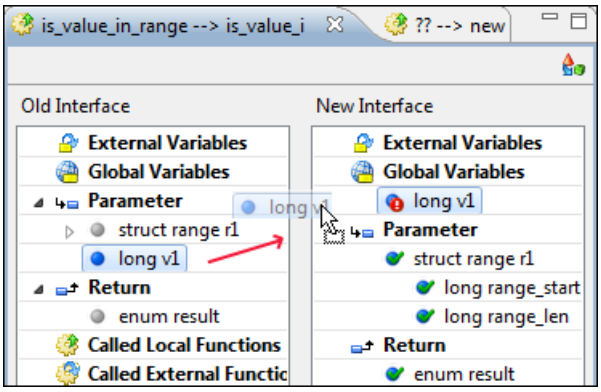


Figure 6.104: Use drag and drop in IDA

To commit assignments:

→ Click on (Commit) in the menu bar of the Compare view.

The data of all test cases and test steps will be copied from the old interface to the current test object interface.

The test object changes to yellow to indicate that all test cases are ready to be executed again.

Please notice the following habits:

- Removed and changed test objects require a reuse operation before you can further operate on them.
- Those test objects that remained unchanged will automatically be reused, e.g. they will be ready to use without further activities required.
- Removed test objects will only be displayed as “removed”, if they did contain any test cases and test steps.

6.10 SCE: Component testing

The component test feature is only used for integration testing. You do not need this feature for unit testing.

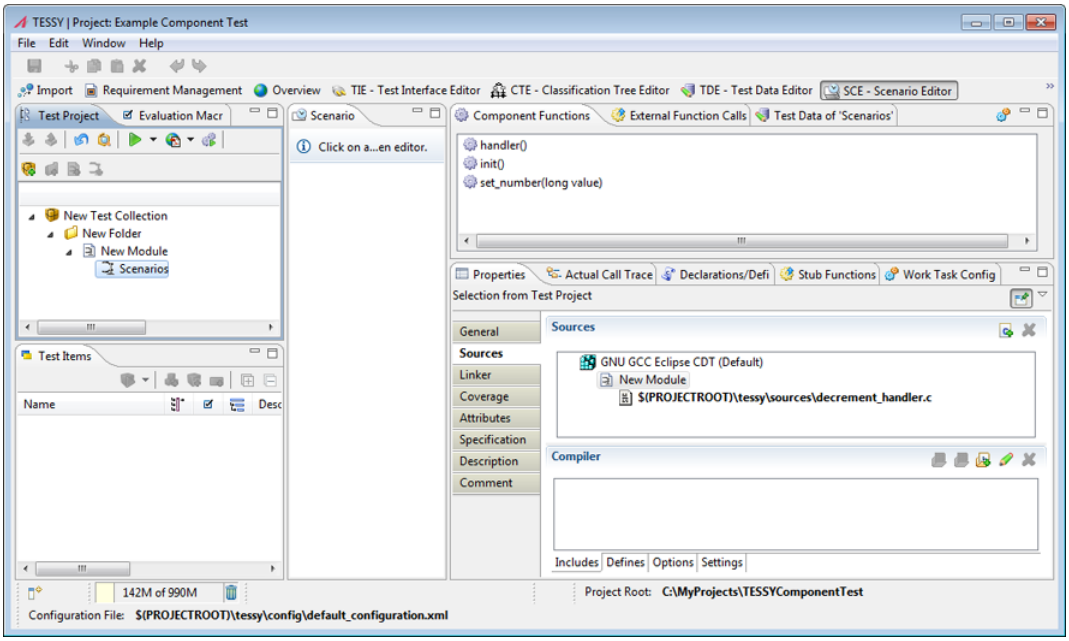


Figure 6.105: Perspective SCE - Scenario Editor

The component test feature within TESSY supports testing of several functions (representing the software component) that interact with themselves as well as with underlying called functions (of other components). The main difference to unit testing of individual functions is the focus of testing on the external interface of the component instead of internal variables or control flow. You should be familiar with the overall usage of TESSY for the normal unit testing. Some features like usercode and stub functions are still available for component testing, but the respective editors will be found at different locations.

The component test feature allows creating calling scenarios of functions provided by a software component. Within these scenarios, the internal values of component variables and any calls to underlying software functions can be checked. TESSY provides the Scenario Editor (SCE) for this purpose. All scenario-related inputs are available through the SCE. Instead of having individual test objects and test cases for the component functions, the component test itself provides a special node scenarios seen as one test

object. The test cases belonging to the scenarios node are the different scenarios for the component.

Within one scenario, you can set global input variables, call component functions, check the calling sequence of underlying software functions and check global output variables. The content of each scenario may be divided into the following parts:

- setting the input variables
- calling component functions
- checking calls to underlying functions
- setting/checking variables during scenario execution
- executing usercode and eval macros
- checking the output variables

The Usercode Editor (UCE) is not available for component testing, because the prolog/epilog code and definitions/declarations sections can be edited directly within the SCE. You will find C-Code-Fragments that can be added into the scenario control flow. Also the code for stub functions may be edited directly within SCE.

6.10.1 Creating component tests

The component test management is based on the TESSY modules.

→ Create a new module as described in section 6.1.2 Creating tests.

→ Include all the source files, include paths and defines necessary to analyze the source code of the component.

→ Activate "component" as kind of test (see figure 6.106).

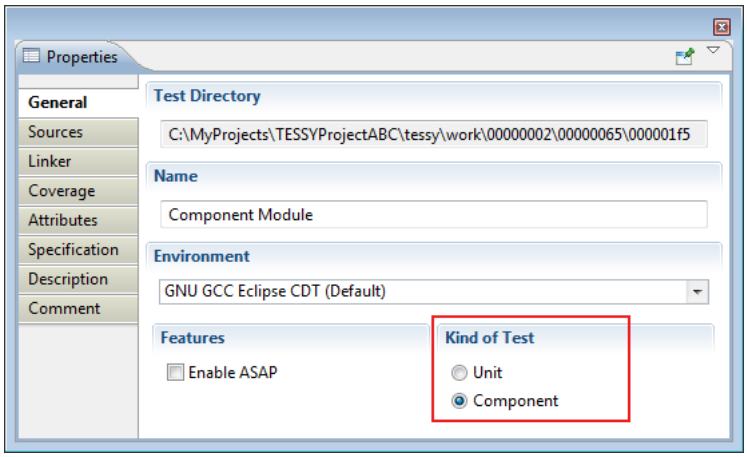


Figure 6.106: Component test

In contrast to normal unit tests, you will only see one special test object called "Scenarios".

6.10.2 Preparing the test interface

The interface of the component is a summarized interface of all the non-static component functions:

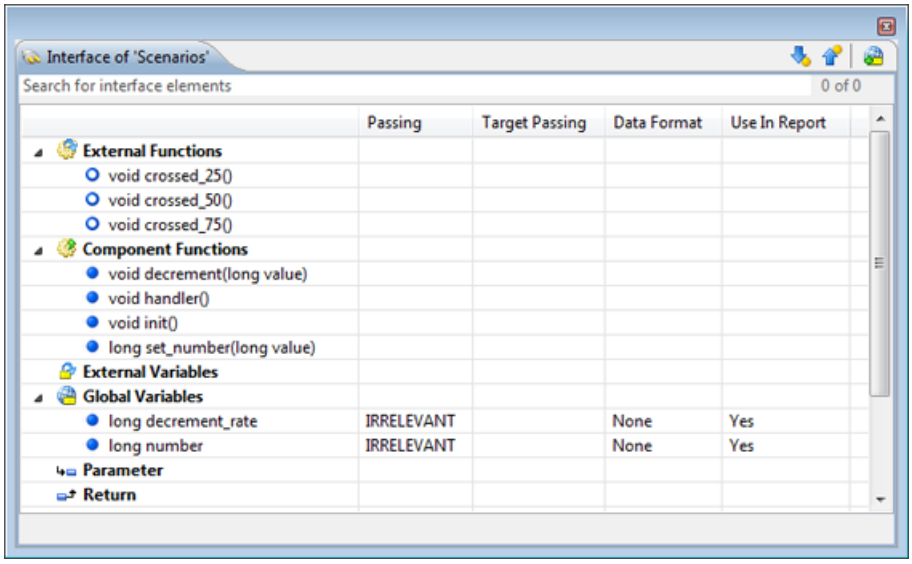


Figure 6.107: Interface of the scenatios

The External Functions section marked with the icon lists the interface to the underlying software functions, if any external function is called from the component. These

external functions can be replaced by stub functions like within the normal unit test. The Component Functions section marked with the icon lists all the component functions, i.e. these are the functions visible from outside the component. Local static functions will not be listed here. The meaning of the status indicators for component functions is as follows:

status indicator	**status / meaning**
	Function is not used for component test.
	The variables used by this function are not available within the component test interface of the scenario. These variables are set to IRRELEVANT and the passing direction cannot be changed.
	The variables used by this function will be available within the scenario and the passing direction may be adjusted.

Table 6.189: status indicators of the Interface view of a component test

6.10.3 Designing the test cases

Testing a component requires a set of scenarios that stimulate the component and check the behavior of the component. Such a scenario contains calls to component functions and other possible actions and expected reactions of the component. A scenario can be seen as a test case for the component. Therefore, Tessy provides a scenario test case list for the component within the right side of the main TESSY window. The term scenario will be used as a synonym for scenario test case in the scope of the component test.

There are two possibilities for creating scenarios: By creation ad hoc or by developing them systematically using the classification tree method supported by CTE within TESSY.

After synchronizing the CTE test cases there will be the respective number of scenarios within TESSY. You can add additional scenarios using the context menu within the scenario list. To edit the scenario, start the scenario editor SCE. The (empty) scenarios will be displayed within SCE providing the specification and description of the designed scenario test cases.

After implementing the scenarios within SCE, execute the scenarios:

→ Select the desired scenario test cases and execute the test using the Execute Test button within the toolbar.

6.11 Backup, restore, version control

With TESSY you can easily backup modules into a directory and check in into a version control system. Modules can also be restored from that directory which facilitates checking out modules from the version control system onto another computer and restoring the test database. You can backup individual modules, folders or whole test collections. The backups will be stored as tmb-files. Restoring the files is either possible within the original folder or as well into another location.

6.11.1 Backup

→ Select the module.

→ Select "File" > "Database Backup" > "Save...".

The Save Database dialog will be opened with your module already selected. The Output Folder contains the backup directory of the current project (see figure 6.108). We recommend to use this directory for any backup and restore operations.
If you have linked your test cases with any requirement documents, you can choose to save the referenced requirement documents as well. This action will create XML files containing the exported requirement documents that are linked with tests of the current module.

→ Click "OK".

Backup

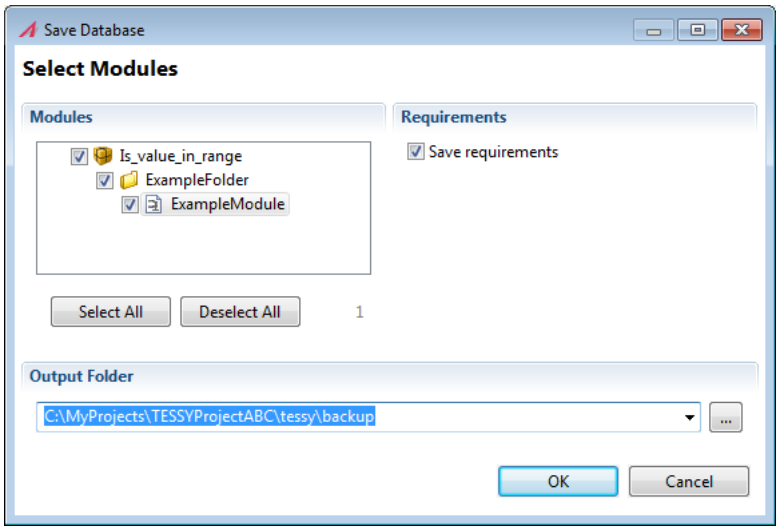


Figure 6.108: Save Database dialog

6.11.2 Restore

Restore into the original location

→ Select a test collection or folder.

→ Select "File" > "Database Backup" > "Restore...".

The Restore Database dialog will be opened with your module already selected. The Output Folder contains the backup directory of the current project. We recommend to use this directory for any backup and restore operations (see figure 6.109).

→ If there are any requirement document backups, the respective requirement documents will appear within the dialog.

→ Click "OK".

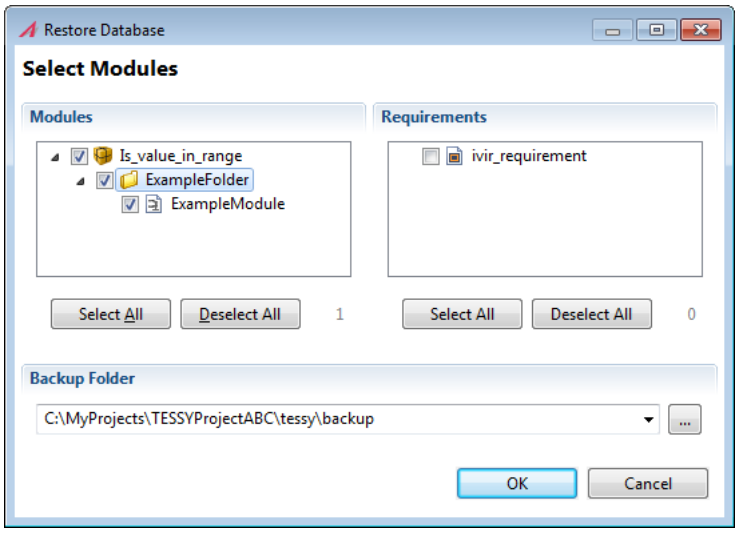


Figure 6.109: Restore Database dialog

Restore into another location

You can also restore TMB backup files into another than the original location: If you select any folder for which there are no corresponding TMB backup files, restore any of the available TMB files as children of this folder. The original test collections and folders of the TMB files will be restored as sub folders of the current folder instead.

6.11.3 Version control

We recommend to save backups of all test relevant files into a version control system on a regular basis. At least when the test development is completed, the whole test setup should be saved as backup.

Version Control

→ Follow the steps described above to create the necessary files.

→ Save the following files and contents of directories into you version control system:

PDBX file
Contents of the ***config*** directory
Contents of the ***backup*** directory

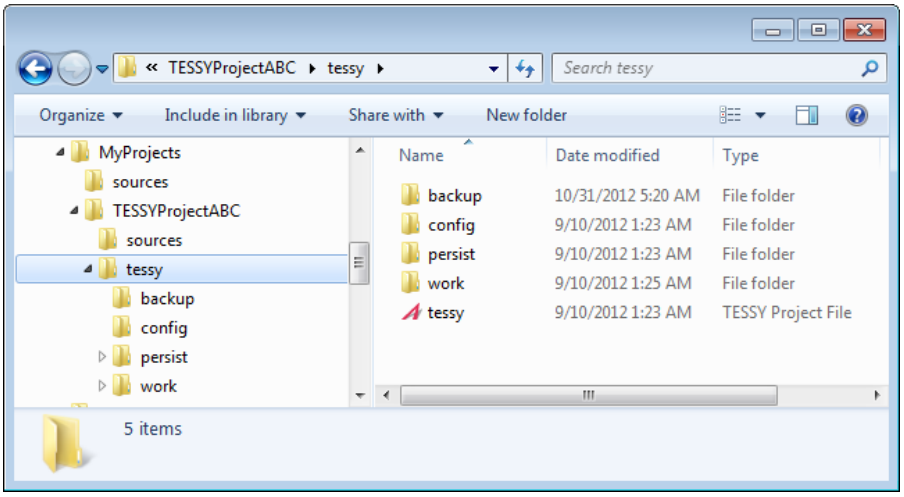


Figure 6.110: Data and files within the project directory

The directory ***work*** contains only temporary files created during development and execution of the tests. You can delete this complete directory to save disk space after the testing work is completed.

The directory ***persist*** contains the current databases of the test project. This directory and the sub directories will be restored when restoring TMB backup files. The valuable contents of this directory will be saved into the TMB files created during the backup process.
When you restore the whole project onto another computer, the directory ***persist*** will be restored from the TMB backup files.

6.12 Command line interface

TESSY provides a command line interface which allows writing batch script files that control various operations within a TESSY project. The command line operations are available by invoking an executable program called "tessycmd.exe".
The program can be called either from a DOS command line shell or from a DOS batch script or other script formats that support calling DOS executables.

Important: You need to start TESSY before invoking any tessycmd.exe commands. tessycmd.exe will connect to a running instance of TESSY in order to execute any commands.

6.12.1 Invoking tessycmd.exe

The executable that provides all command line operations is available within the TESSY installation directory: `Program Files\Razorcat\TESSY_3.0\bin\tessycmd.exe`

- → Start a DOS shell.
- → Change to the directory "bin".
- → Call the executable.

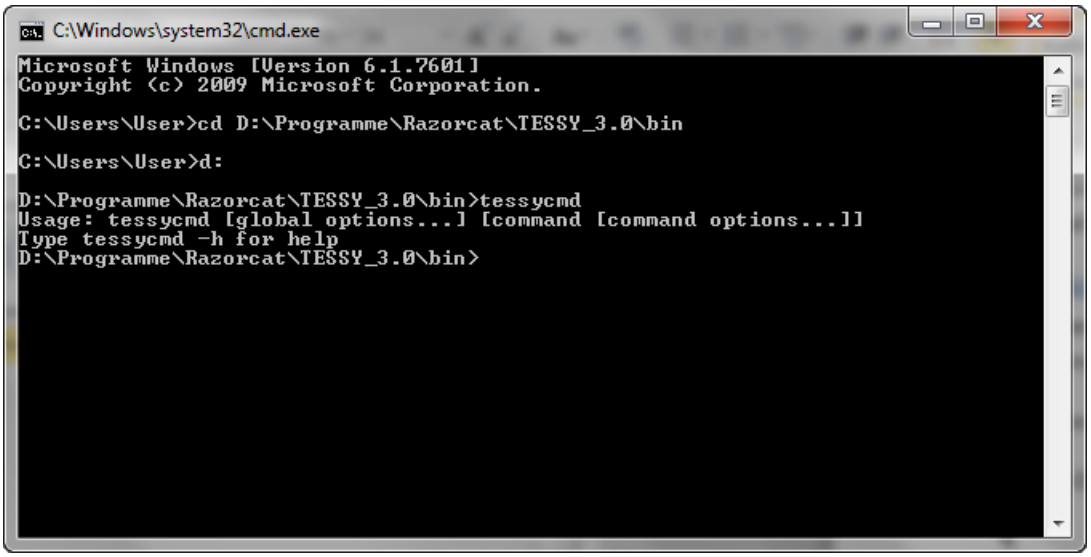


Figure 6.111: DOS command line shell

6.12.2 Usage of tessycmd.exe

The available commands provide means to create, select and list TESSY objects, i.e. a project, test collection, folder, module, test object. After invoking any create commands, the respective new TESSY object will be selected. You can invoke further commands to manipulate any previously created or selected TESSY objects.

You need to call all commands according to the following sequence:

- Connect to TESSY.
- Select or create TESSY objects.
- Invoke commands to start operations on the selected TESSY objects.
- Disconnect from TESSY.

Important: If you are not connected, invoking any commands will fail. Multiple connects will also cause an error.

The current state (connection and selection of TESSY objects) of the tessycmd.exe executable is managed by the currently running TESSY application. If you restart TESSY, the state of tessycmd.exe will be reset to the initial state, i.e. disconnected.

6.12.3 Commands

To geht a complete list of the available commands:

→ Enter ***tessycmd -h***.

To connect the running instance:

→ Enter ***tessycmd connect***.

To list the available projects:

→ Enter ***tessycmd list-projects***.

To disconnect from TESSY:

→ Enter ***tessycmd disconnect***.

To execute tessycmd.exe within any directory, add the directory "bin" of the TESSY installation to the windows path environment variable.

6.12.4 Example: DOS script

You will find the following example DOS script within the TESSY installation directory:

`Program Files\Razorcat\TESSY_3.0\Examples\CommandLine\cmd_line_example.bat`

The script is prepared to import TESSY backup files (TMB-files) into the currently open TESSY project. It will create a new test collection "Examples" and import the existing TMB files into a newly created folder. After the import, it executes the imported modules:

→ Start TESSY, create a new project and open this project.

→ Start a DOS command shell.

→ Change to the bin directory of the TESSY installation:
`C:\Program Files\Razorcat\TESSY_3.0\bin`

→ Invoke the DOS batch script "cmd_line_example.bat". Alternatively, double-click on the file "cmd_line_example.bat" which invokes the DOS script within a new DOS shell. There is a pause command at the end which causes the DOS shell to remain open after execution of the script.

Appendix

A Abbreviations

API	Application Programming Interface
CTE	Classification Tree Editor
CV	Coverage Viewer
DOS	Disk Operating System
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
IDA	Interface Data Assigner
IEC	International Electrotechnical Commission
ISO	International Standards Organization
PDB	Project Database
QTS	Qualification Test Suite
RQMT	Requirement
SCE	Scenario Editor
TDB	Test Database
TDE	Test Data Editor
TEE	Test Environment Editor
TIE	Test Interface Editor
TMB	Timbuktu Pro Connection Document
XML	Extensible Markup Language

B Glossary

Batch Testing A testing procedure in which multiple test objects are executed automatically one after each other without further user interaction.

Branch Coverage Is usually abbreviated "C1". Roughly spoken: Branch Coverage reveals, if all branches were executed, for example, an if-instruction has two branches, the then-branch and the else-branch.

C1 Test During a C1 test, each branch of the test object will be instrumented with a counter to monitor, how often a branch of the program is run through.

Classification Tree The objective of the Classification Tree Method is to determine a sufficient but minimum number of test cases. It is a systematic approach to test planning by test case specifications and priorizations.

Code Coverage A test object is considered to consist of items like branches, conditions, etc. Code coverage measures, how many of the items were exercised during the tests. This number is related to the total number of items and is usually expressed in percent. TESSY features C1 coverage (branch coverage) and C2 coverage (MC/DC: Modified Condition)

Component Testing is the test of interacting test objects, i.e. interacting functions in the sense of C. These functions can be a (single) calling hierarchy of functions, but we will consider this mainly as unit testing. We consider as a component mainly a set of functions that interact e.g. on common data and do not necessarily call each other. Component testing then is testing of such a set of functions. The units do not need to be in a calling hierarchy; they may only interact on data, like push() and pop() of the abstract data type "stack". A component according to this specification may also be called a "module", and its testing "module testing" respectively.

Debugger A computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered but which will typically be somewhat slower than executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operation - full or partial simulation, to limit this impact.

Enums Type of the C language specification which allows to define a list of aliases which represent integer numbers.

Expected Values Values expected to be calculated by the test object. The result values are checked against the expected values after the test run.

Hysteresis Dependence of a system not just on its current environment but also on its past. This dependence arises because the system can be in more than one internal state.

Interface Data Assign editor (IDA) If the interface elements of the test object have changed, you can assign the new interface elements to the old. Your test data will be assigned automatically.

Input Values Function parameters, global and external variables which have effect on the behavior of the function.

Interface Description Information about the passing direction and type of interface elements (parameter, global variables and extern variables). TThe interface description is determined automatically by TESSY and is made visible / changeable in the TIE.

Integration Testing can be considered either as unit testing for a calling hierarchy of functions or as a component testing for a set of interacting functions not necessarily calling each other. Component testing is integration testing of the functions in the component.

Modified Condition / Decision Coverage (MC/DC) MC/DC coverage takes the structure of a decision into account. Each decision is made up from conditions, which are combined by logical operators (and, or, not). Roughly spoken, to get 100 percent MC/DC for a decision, each condition in the decision requires a pair of test cases, that

- differs in the Boolean value for that condition, and
- has the same boolean value for all other conditions, and
- produces true and false in the outcome of the whole decision

n conditions require n+1 test cases

Module A TESSY module comprises primarily of the test object (in C a function in the sense of C) and source files, compiler settings, interface description and test data. You can pool modules in projects.

Output Values The same as an expected value in the TESSY context. Both terms are used in equivalence within this manual. The output (repectively expected) values are evaluated against the actual result values after the test run.

Project Root See explanatory box on page 22.

Regression Testing Regression testing is the repetitive running of already successfully completed test cases. The intention of regression testing is to verify that modifications and enhancements to a test object do not break the already successfully completed tests.

Requirement Documented need of what a test should perform and important input for the verification process. Requirements show what elements and functions are necessary for the test.

Requirement Management Management of different types of requirements that need to be covered by at least one test.

Requirement, Functional Describes the features, specific behavior, business rules and general functionality that the proposed system must support.

Requirement, Non-Functional Specifies criteria that can be used to judge the operation of the test.

Search Query Search definitions processed by a search engine.

Stubs piece of code used to stand in for some other programming functionality. A stub may simulate the behavior of existing code (such as a procedure on a remote machine) or be a temporary substitute for yet-to-be-developed code.

System Testing is the testing of the application (software or software and hardware) as a whole.

Test Data Editor (TDE) With the TDE you can enter the input values and expected values for the test run.

TESSY Support File Contains information about test objects including data, compiler, project settings etc. It helps the support to detect the cause of a problem. In section Troubleshooting it is explained how to create a TESSY Support File.

Test Case Element that encapsulates the abstract test definition, e.g. the specification and description of a test, and the concrete test data managed within test steps.

Test Definition Describes a test to be performed on the test system in textual format. A test definition abstractly describes the inputs and the expected outcome of a test and refers to a list of requirements which shall be validated with this test.

Test Driver C-source files generated by TESSY for the test execution. These files are compiled and linked in order to build an application that prepares the input data, call the test object and store the actual result data.

Test Environment Information about the test object, the compiler used, the target debugger or emulator and more settings.

Test Object The function to be tested.

Test Run One execution of a test object with the given test cases. The result of a test run is stored within an XML result file that may be further processed by external tools.

Test Suite A collection of test objects with test scenarios and/or test cases that were created to fulfill a certain test objective.

Test Interface Editor (TIE) With the TIE you can view all interface elements and review or set the passing direction and/or other information of the interface elements.

Usercode In the usercode you can enter C code, which is executed before or after test cases/test steps during the execution of a test object.

Workspace The space at local disk where the TESSY application reads and writes data. Place for configuration and temporary report data. Project data can be saved separately.

C List of Figures

0.1 Core workflow of TESSY . . . xxi
0.2 New Test Project view . . . xxiv
0.3 New traceability of requirements . . . xxv
0.4 Defining coverage measurements . . . xxv
0.5 New tabular Test Data Editor . . . xxvi
0.6 New Test Data view . . . xxvii
0.7 Dialog for creating the TESSY Support File . . . xxix
0.8 Dialog for logging settings . . . xxx

1.1 InstallAware Wizard . . . 2
1.2 License Agreement . . . 2
1.3 Setup Type . . . 3
1.4 Destination Folder . . . 3
1.5 TESSY Testarea Folder . . . 4
1.6 Program folder and user selection . . . 4
1.7 Installing process of TESSY . . . 5
1.8 Installation is completed . . . 5
1.9 License Key Request . . . 6
1.10 Configure menu of the license server . . . 7
1.11 License key check successful: this license key is correct . . . 8
1.12 License key check unsuccessful: license key is incorrect for the host id . . . 9

3.1 Loading TESSY . . . 19
3.2 Path of the workspace . . . 19
3.3 Selecting a folder for the project root. . . . 20
3.4 Creating a new project . . . 21
3.5 Project Example1 is created. . . . 21
3.6 Deleting a project . . . 24
3.7 TESSY interface and its terminology . . . 25
3.8 Open other perspectives . . . 26
3.9 Test Project view within the Overview perspective . . . 27
3.10 Adding views to a perspective . . . 28
3.11 Move the views separately. To switch back, use “Reset”. . . . 28
3.12 To switch back all positions of views and perspectives use “Reset Workbench”. . . . 29
3.13 Minimizing and maximizing views . . . 29
3.14 Maximized view with minimized views on the right and the restore-button on the left . . . 30
3.15 Using the context menu with right click. . . . 31

4.1 Initial equivalence partitioning for “ice warning” . . . 44

4.2 Repeated equivalence partitioning for "ice warning" . . . 45
4.3 A possible CT for "ice warning" . . . 46
4.4 Result of the CTM: tree (above) with combination table (below) . . . 47
4.5 The problem "is_value_in_range" depicted graphically . . . 48
4.6 The initial CT with three test-relevant aspects . . . 48
4.7 The CT for is_value_in_range, 2nd step . . . 49
4.8 The CT for is_value_in_range, 3rd step . . . 49
4.9 A first specification for the range in the combination table . . . 50
4.10 A second specification for the range in the combination table . . . 50
4.11 The CT for is_value_in_range, 4th step . . . 51
4.12 The third range specification provokes a wrap-around . . . 52
4.13 The completed CT for is_value_in_range . . . 53
4.14 The completed test case specification . . . 55
4.15 An alternative test case specification . . . 56

5.1 Operational sequences in TESSY . . . 61
5.2 Creating the new project "Example1". . . . 63
5.3 Test collection "Is_value_in_range" with an example folder and module . . . 64
5.4 GNU GCC compiler is selected by default. . . . 64
5.5 The source code of the C-Function to be tested . . . 65
5.6 Adding the C-source file. . . . 66
5.7 Analyzing the module, that is the C-source file. . . . 67
5.8 The function of the C-source is displayed as child of the module. . . . 67
5.9 Perspective TIE - Test Interface Editor . . . 68
5.10 The inputs and outputs are already defined . . . 69
5.11 Test Items view . . . 70
5.12 Three test cases were added in the Test Items view . . . 70
5.13 Data is entered, test step turns yellow and test case is ready to run. . . . 72
5.14 Entering data for test object is_value_in_range . . . 73
5.15 The test cases are ready to test . . . 73
5.16 TDE after test run is_value_in_range . . . 74
5.17 Test results of is_value_in_range . . . 75
5.18 Selecting Branch and MC/DC Coverage for test run . . . 76
5.19 Execute Test dialog while running the test . . . 76
5.20 Branch coverage is_value_in_range . . . 77
5.21 Decision coverage is_value_in_range . . . 78
5.22 Code section of the if branch of the first decision . . . 79
5.23 Code section of the second decision . . . 80
5.24 Creating the folder for reports . . . 81
5.25 Content of the test report is_value_in_range . . . 82
5.26 Requirement Management perspective . . . 83
5.27 Import perspective of the requirement management . . . 84
5.28 The new requirement document . . . 84
5.29 Entering an alias for the requirement document . . . 85
5.30 Comment for the initial revision of the commit . . . 85
5.31 Linking test cases with requirements . . . 86
5.32 Test Definition view within TDE with linked requirement . . . 87
5.33 Editing the settings of a Planning Coverage Report . . . 88

5.34 Dialog of the settings for the Planning Coverage Report . . . 88
5.35 Planning coverage report of the IVIR requirement document . . . 89
5.36 Generating a Test Details Report . . . 90
5.37 Part of the generated test report of is_value_in_range . . . 91
5.38 Creating an Execution Coverage Report . . . 91
5.39 Coverage Report of is_value_in_range . . . 92
5.40 Overview perspective after test run (with requirements) . . . 94
5.41 Use the context menu to edit a source . . . 94
5.42 Editing the C-source file is_val_in_range.c . . . 95
5.43 Changed C-source file of is_value_in_range . . . 95
5.44 Adding a "delete" and "new" object . . . 96
5.45 Changed and new test objects of is_value_in_range . . . 96
5.46 Remove the code for test object "deleted". . . . 97
5.47 Changed and new test objects of is_value_in_range . . . 98
5.48 Changed, deleted and new test object of is_value_in_range . . . 99
5.49 Use drag and drop in IDA . . . 100
5.50 CTE with the root "is_value_in_range" . . . 101
5.51 Creating a new classification with the context menu . . . 102
5.52 Renaming the new classification . . . 103
5.53 Classifications of is_value_in_range . . . 103
5.54 Simple classification tree for is_value_in_range . . . 104
5.55 Entering data in CTE . . . 105
5.56 Entering more data in CTE . . . 106
5.57 Blue dots indicate that "range_start" elements contain data . . . 107
5.58 Creating test cases in the test item list . . . 107
5.59 Defining test cases in the combination table of CTE . . . 108
5.60 Completed table with all test cases for example "is_value_in_range" . . . 109
5.61 Test data is displayed when selecting a test case in the combination table . . . 110
5.62 Test data displayed within TDE . . . 111

6.1 Overview perspective . . . 117
6.2 Test Project view within the Overview perspective . . . 118
6.3 The function of the C-source is displayed as child of the module. . . . 121
6.4 Test Execution Settings . . . 123
6.5 Coverage displayed within the Test Project view . . . 125
6.6 Creating a report . . . 128
6.7 Changing the logo image for reports . . . 130
6.8 Defining the batch operation . . . 130
6.9 Defining the batch operation . . . 131
6.10 Defining the batch operation . . . 131
6.11 Editing the settings of each batch operation . . . 132
6.12 Import settings of data import . . . 133
6.13 Export settings of data export . . . 134
6.14 Properties view . . . 134
6.15 The Sources pane in the Sources tab of the Properties view . . . 136
6.16 The Compiler pane in the Sources tab of the Properties view . . . 137
6.17 The Setting tab of the Properties view with module selected . . . 139
6.18 The Setting tab of the Properties view with C-source file selected . . . 140

6.19 The Coverage tab of the Properties view . . . 140
6.20 The Attributes tab of the Properties view . . . 141
6.21 Creating a new attribute . . . 142
6.22 Requirements Coverage view . . . 143
6.23 Test Items view . . . 143
6.24 First test case with one test step. . . . 145
6.25 Test cases created by CTE . . . 146
6.26 Test Results view . . . 147
6.27 Evaluation Macros view . . . 147
6.28 Console view . . . 148
6.29 Preference "Show console on error" . . . 149
6.30 Import perspective . . . 150
6.31 Adding files in the Import perspective . . . 151
6.32 Requirement Management perspective . . . 154
6.33 RQMT Explorer view . . . 155
6.34 The document is modified, because the alias was modified . . . 157
6.35 Committing options . . . 157
6.36 Comment for the initial revision of the commit . . . 158
6.37 After the commit . . . 158
6.38 Double click on the RQMT opens the requirement editor . . . 159
6.39 The first requirement was modified . . . 159
6.40 The first requirement has the version 2.0 . . . 160
6.41 Requirements List view . . . 160
6.42 Requirements List view with test and a figure . . . 161
6.43 Test Means view . . . 162
6.44 VxV Matrix view . . . 163
6.45 Link Matrix view . . . 164
6.46 Adding Test Cases to Link Matrix view . . . 165
6.47 Link Matrix view with suspicious elements . . . 166
6.48 Suspicious Elements view . . . 167
6.49 Suspicious test object and test cases . . . 168
6.50 Suspicious test object and linked modified requirements . . . 168
6.51 Selecting the suspicious test case shows the modified requirement(s) . . . 169
6.52 Comparing the versions of the requirement . . . 169
6.53 Attached Files view . . . 170
6.54 Attributes view . . . 171
6.55 Editing attribute settings within the Attributes view . . . 172
6.56 History view . . . 173
6.57 Differences view . . . 174
6.58 Related Elements view . . . 175
6.59 Requirements Coverage view . . . 175
6.60 Options of refreshing . . . 176
6.61 Test Environment Editor (TEE) . . . 179
6.62 Modes of TEE to display the settings . . . 181
6.63 Environments of the Eclipse target . . . 181
6.64 Enabled and disabled configurations within TEE . . . 186
6.65 Enable compiler . . . 187

6.66 Perspective TIE - Test Interface Editor . . . 191
6.67 Information of passing direction and type . . . 193
6.68 White arrow indicating further levels, black arrow when expanded . . . 193
6.69 Resetting passing directions . . . 197
6.70 Setting the data format . . . 197
6.71 Create a stub function within the context menu . . . 199
6.72 CTE perspective . . . 202
6.73 Classification Tree view . . . 204
6.74 Creating a new classification with the context menu . . . 206
6.75 Creating test cases in the test item list . . . 208
6.76 TDE perspective . . . 210
6.77 Type information of the variable long range_start . . . 212
6.78 Test Data view . . . 213
6.79 Test step 1.1 is selected and undefined values are highlighted in yellow . . . 215
6.80 Clicking in the cell opens the available enums . . . 218
6.81 Entering evaluation mode "unequal" within the inline editor . . . 221
6.82 Test Definition view within TDE with linked requirement . . . 224
6.83 Prolog/Epilog view . . . 225
6.84 Call sequence of the usercode parts . . . 225
6.85 Prolog/Epilog functions . . . 226
6.86 Editing C code . . . 227
6.87 Call the popup menu by pressing CTRL + space . . . 228
6.88 Usercode Outline view . . . 230
6.89 Plots view . . . 231
6.90 Creating a new plot with the context menu . . . 232
6.91 Creating a new plot . . . 232
6.92 Adding variables to a plot . . . 233
6.93 Plots view with a plot and two variables . . . 234
6.94 Uncheck variables for a better overview . . . 235
6.95 Highlighted upper value for variable sine_value . . . 235
6.96 Perspective CV - Coverage Viewer . . . 236
6.97 Selecting Branch and MC/DC Coverage for test run . . . 237
6.98 Flow Chart view . . . 239
6.99 Unreached connection is marked blue . . . 242
6.100Branch coverage . . . 243
6.101Decision coverage . . . 244
6.102IDA perspective . . . 246
6.103Compare view . . . 248
6.104Use drag and drop in IDA . . . 249
6.105Perspective SCE - Scenario Editor . . . 251
6.106Component test . . . 253
6.107Interface of the scenatios . . . 253
6.108Save Database dialog . . . 255
6.109Restore Database dialog . . . 256
6.110Data and files within the project directory . . . 257
6.111DOS command line shell . . . 258

D List of Tables

0.1 Where to find - matters of the several parts of the TESSY manual xviii
0.2 Font characters xx
0.13 Information to log and add to TESSY Support File xxxi

2.1 Changes in handling with TESSY 2.9 and TESSY 3.0 13
2.2 How to import previous projects in TESSY 3.0 15

3.1 File system and databases of TESSY 18
3.17 Shortcuts and key functions 32

5.17 Entering data for test object is_value_in_range. 72

6.1 Structure of the Overview perspective 118
6.2 Tool bar icons of the Test Project view 119
6.3 Status indicators of the Test Project view 120
6.11 Coverage measurements 124
6.15 Reports available within TESSY 127
6.18 Available settings of the reports 129
6.27 General tab of Properties view 135
6.34 Options of module settings in Properties view 139
6.38 Tool bar icons of the Test Items view 144
6.39 Status indicators of the Test Items view 145
6.47 Committing options to control the versioning of imported requirements 151
6.50 Possible formats of requirement sources 152
6.52 Structure of the Requirement Management perspective 155
6.53 Tool bar icons of the RQMT Explorer view 156
6.54 Status indicators of the RQMT Explorer view 156
6.63 Tool bar icons of the Requirements List view 160
6.66 Tool bar icons of the Test Means view 162
6.67 Tool bar icons of the VxV Matrix view 163
6.68 Tool bar icons of the Link Matrix view 164
6.69 Status indicators of the Suspicious Elements view 164
6.75 Tool bar icons of the Suspicious Elements view 167
6.78 Tool bar icons of the Attached Files view 170
6.79 Tool bar icons of the Attributes view 171
6.80 Tool bar icons of the Attributes view 172
6.81 Tool bar icons of the Histroy view 173
6.83 Tool bar icons of the Requirements Coverage view 176
6.86 Selected elements within Requirement Management and displayed contents . . . 177
6.89 Structure of TEE 180

6.94 Status indicators of TEE . . . 183
6.97 Contents, functions and storage location of configuration files . . . 185
6.105Means of flags in the attribute properties . . . 190
6.106Structure of TIE . . . 192
6.107Tool bar icons of the Interface view . . . 194
6.112Possible passing directions of the interface elements . . . 196
6.124Structure of the CTE perspective . . . 203
6.125Tool bar icons of the Classification Tree view . . . 205
6.126Structure of Classification Tree view . . . 205
6.137Structure of TDE . . . 211
6.138Tool bar icons of the Test Data view . . . 214
6.139Interface elements and icons of the Test Data view . . . 214
6.151Options of initializing values . . . 219
6.152Initialization values for data types . . . 220
6.154Evaluation modes . . . 221
6.157Value assignments for pointers . . . 223
6.164Available types of evaluation macros . . . 229
6.165operators of evaluation macros . . . 229
6.167Tool bar icons of the Plots view . . . 231
6.174Structure of CV . . . 238
6.175Tool bar icons of the Test Items view . . . 240
6.177Elements of the Flow Chart view . . . 241
6.181Structure of the IDA perspective . . . 247
6.182Status indicators of the Compare view . . . 248
6.189status indicators of the Interface view of a component test . . . 254

Index

Array Plot, 231
Assigning changed Interface, 98
Attributes
    Adding, 188
    Editing, 188

Backup, 255
Batch Operation, 130
Batch Test, 130, 264
Branch Coverage, 264
Branch Coverage View, 77, 243

C Code
    Editing, 227
    Entering, 226
C-source file
    Analyzing the, 65
    Editing the, 94
C1 Test, 264
Classes
    Creating, 206
Classicication Tree Editor, 202
Classification Tree, 264
Classification Tree View View, 204
Classifications
    Creating, 206
Code Coverage, 264
Command Line Interface, 258
Commit Changes, 151
Committing Requirements, 157
Compare View, 99, 248
Compiler
    GNU GCC, 64
Component Testing, 264
Config File, 18
Configuration File, 18, 23, 184
    Creating a, 22
Console View, 148
Context Menu, 31
Coverage
    Instrumentation, 237
    Selecting Measurement, 237
Coverage Viewer, 76, 236
CTE, 202
CV, 76, 236

Data Format
    Setting the, 197
Debugger, 264
Delete
    Compiler, 182
    Environment, 182
    Folder, 31
    Module, 31
    Project, 23
    Target, 182
    Test Case, 31
Differences View, 169, 174

Edit Source, 94
Enable Configurations, 186
Entering Plots, 231
Enums, 217, 264
Eval Macros, 228
Eval Modes, 220
Evaluation Macros, 228
Evaluation Macros View, 147
Evaluation Modes, 220
Executing a Test, 122
Expected Values, 264

File System, 18
Flow Chart View, 77
Formats of Requirements, 152

GNU GCC Compiler, 64

Hysteresis, 265

IDA, 93, 246, 265
Import, 150

Importing Requirements, 83
Include Path, 137
Inheritance of attributes, 181
Inline Mode, 217
Input Values, 265
Instrumentation Settings, 124
Integration Testing, 265
Interface Changes, 93
Interface Data Assign Editor, 93, 246
Interface Description, 265

License Key File, 6
License Key Request, 6
Link Matrix view, 164

Maximum Criterion, 47
MC/DC, 265
MC/DC Coverage View, 77, 244
Menu Bar, 25
Minimum Criterion, 47
Modified Condition / Decision Coverage, 265
Module, 63, 265

Output Values, 265
Overview Perspective, 63, 117

PDB-File, 18
Perspective, 26
    CTE, 202
    CV, 76, 236
    IDA, 93, 246
    Import, 150
    Overview, 63, 117
    Requirement Management, 83, 154
    SCE, 251
    TDE, 71, 210
    TEE, 64, 179
    TIE, 68, 191
Perspective Bar, 26
Plots
    Entering, 231
    Setting Properties, 232
Plots View, 231
Plots view, 211
Pointers, 198, 222
Project Database, 18
Project Root, 18, 65, 66
Prolog/Epilog View, 225
Properties View, **134**, 192, 212

Registration, 6
Regression Testing, 93, **246**, 265
Rename, 31
Repeat Count, 225
Report
    Changing Logo, 129
    Creating a, 81, 126
    Execution Coverage, 126
    Planning Coverage, 87, 126
    Test Details, 81, 126
    Test Overview, 126
Requirement, 265
    Alias, 156
    Committing, 157
    Coverage, 175
    Creating, 155
    Document, 151
    Engineering, 155
    Formats, 152
    Functional, 266
    Import, 150
    Linking of, 175
    Management, 153, 266
    Non-Functional, 266
    Renaming Document, 156
    Sources, 152
    Versioning of, 151, 173
Requirement Management, 83, 153
Requirements
    Editing, 159
    Importing, 83
Requirements List View, 160
Restore, 255
Reuse, 93, 246
RQMT Explorer View, 84, 155
Running a Test, 122

Saving Changes, 26
SCE, 251
Scenario Editor, 251
Search Query, 266
Shortcuts, 31
Source
    Analyzing the, 65
    Editing the, 94
Status Bar, 30
Stub Functions, 198
Stubs, 266

Support, xxviii
Support File, xxviii
Suspicious Elements, 166, 167
System Testing, 266

Target Passing, 198
TDE, 71, **210**, 266
TEE, 64, 179
TESSY Support File, xxviii
Test Case, 266
    Plot, 231
Test Data
    Exporting, 132
    Importing, 132
Test Data Editor, 71, **210**, 266
Test Data View, 211, 213
Test Definition, 266
Test Definition View, 87, 224
Test Driver, 266
Test Environment, 266
Test Environment Editor, 64, 179
Test Execution, 122
Test Interface Editor, 68, **191**, 266
Test Items View, 69, 143
Test Means view, 162
Test Object, 67, 266
    Changing Interface, 94
    External Functions, 194
    External Variables, 194
    Global Variables, 194
    Local Functions, 194
Test Project View, 63, 118
Test Results View, 147
Test Run, 266
Test Step
    Plot, 231
Test Suite, 266
Testrun, 122
TIE, 68, 191, 266
Tool Bar, 26
Troubleshooting, xxviii
Type Information, 192, 212
Type Modifier, 192, 212

Usercode, **225**, 230, 267
Usercode Outline View, 230

Version Control, 157, 160, 161, 255
View, 27
    Branch Coverage, 77
    Attached Files, 170
    Attributes, 171
    Branch Coverage, 243
    Classification Tree, 204
    Compare, 99, 248
    Console, 148
    Differences, 169, 174
    Evaluation Macros, 147
    Flow Chart, 77
    History, 173
    Interface, 68
    Link Matrix, 164
    MC/DC Coverage, 77, 244
    Plots, 231
    Prolog/Epilog, 225
    Properties, 64, **134**, 192, 212
    Related Elements, 175
    Requirements Coverage, 175
    Requirements List, 160
    Reset Position, 28
    RQMT Explorer, 84, 155
    Suspicious Elements, 167
    Test Data, 71, 213
    Test Definition, 87, 224
    Test Items, 69, 143
    Test Means, 162
    Test Project, 63, 118
    Test Results, 147
    Usercode Outline, 230
    VxV Matrix, 163

Without Commit, 151
Workspace, 18, 267